Semantic Refinements for Program Verification

Satoshi Kura

Dissertation

submitted to the Department of Informatics
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

The Graduate University for Advanced Studies,
SOKENDAI
March 2022

Abstract

Semantics of programming languages is an essential theory not only for defining the meaning of programs but also for studying properties of programs. This thesis aims to apply program semantics to verification of programs. Specifically, we study refinements of semantics of programs to enhance expressivity and verification power. The refined models maintain essential mathematical structures of the original models and have more information about properties of programs. By replacing the original models with the refined models, we obtain improved verification methods. In the thesis, we consider four applications of semantic refinements.

The first one is a semantic construction of dependent refinement type systems. A dependent refinement type system is a type system that admits both refinement types and dependent types. Refinement types are types whose values are restricted by predicates and can be used to specify preconditions and postconditions of functional languages. Dependent types are types that depend on other terms and allow us to use postconditions that depend on the input value. We study categorical semantics of dependent refinement type systems. Our construction is based on the following intuition: a dependent refinement type system is obtained from an underlying type system (a type system that does not contain refinement types) by refining types by predicates. We formalize a semantic counterpart of this intuition. Given a model of the underlying type system (a closed comprehension category and a fibred monad) and a model of predicate logic (a posetal fibration with some conditions), we construct a model of the dependent refinement type system (a closed comprehension category and a fibred monad that refines the model of the underlying type system). We show that we can define the interpretation of refinement types using our construction. We also provide several examples of our construction.

The second one is a program logic for effect handlers. Effect handlers are a programming language feature that allows programmers to implement user-defined computational effects. However, verification of effect handlers is not yet well-studied. We provide a program logic for effect handlers by considering refined semantics of effect handlers. Specifically, we consider Hoare triples for effect handlers and interpret them as liftings of the interpretation of effect handlers along a fibration. We consider sufficient conditions under which we can construct those liftings from liftings of each operation in effect handlers. Such conditions lead to inference rules of our program logic that provide compositional reasoning about effect handlers.

The third one is decision tree-based ranking function synthesis. Ranking functions are an essential notion for termination analysis. We propose an example-based termination analyzer that can synthesize piecewise affine ranking function. Our analyzer finds a piecewise affine ranking function for a given program by repeatedly guessing a candidate solution from a finite set of examples of the transition relation and accepting the genuine solution from the candidates. We refine an existing example-based method that synthesizes affine ranking functions so that our method can synthesize piecewise affine ranking functions. Our method uses decision trees to express affine ranking functions and extends the decision tree learning algorithm for transition examples. Our extended decision tree learning algorithm detects a certain kind of cyclic constraints in transition examples and resolves them by appropriately splitting the state space. We implemented our synthesizer and compared our tool with other tools.

The fourth one is a method for overapproximating tail probabilities of runtime of randomized programs. It is known that ranking supermartingales can give an upper bound of the expected runtime of randomized programs. This fact can be used to overapproximate tail probabilities of runtime of randomized programs by applying concentration inequalities like Markov's inequality. We refine the existing notion of ranking supermartingales so that they can also give upper bounds of higher moments of runtime. Technically, our improvement is based on the order-theoretic characterization of ranking supermartingales. It is known that the expected runtime is the least fixed point of a certain monotone function, and ranking supermartingales are prefixed points of the monotone function. We extend this to characterize higher moments of runtime as the least fixed point and define ranking supermartingales for higher moments as prefixed points. This extension allows us to improve upper bounds of tail probabilities. We implemented a synthesizer of our notion of ranking supermartingales and conducted experiments.

Acknowledgements

I would like to thank my supervisor Ichiro Hasuo for his support of my PhD life and research. My PhD program could not be more fruitful thanks to his guidance, knowledge, and leadership in his great research team. I would like to thank Shin-ya Katsumata, who gave me many helpful comments and advice when I talked about my research progress. This thesis includes joint work with Ichiro Hasuo, Natsuki Urabe, and Hiroshi Unno. I am grateful for fruitful collaborations. Lastly, I would like to thank my parents for their patience and support since my birth and encouraging my decision to enter the PhD program.

Contents

1 Introduction 1

1.1 Backgrounds 1

1.2 Semantic Refinements 3

1.2.1 A Categorical Construction of Dependent Refinement Type Systems 3

1.2.2 A Program Logic for Effect Handlers 5

1.2.3 Decision Tree-Based Ranking Function Synthesis 8

1.2.4 Tail Probabilities of Randomized Programs via Higher Moments of Runtime 11

1.3 Outline of This Thesis and Corresponding Papers 14

2 A Categorical Construction of Dependent Refinement Type Systems 17

2.1 Preliminaries 17

2.1.1 Fibration 17

2.1.2 Comprehension Category 19

2.2 Lifting SCCompCs and Fibred Coproducts 21

2.2.1 Lifting SCCompCs 21

2.2.2 Lifting Fibred Coproducts 37

2.3 Lifting Monads on SCCompCs 40

2.4 Soundness 43

2.4.1 Underlying Type System 43

2.4.2 Predicate Logic 45

2.4.3 Refinement Type System 46

2.4.4 Semantics 48

2.5 Toward Recursion in Refinement Type Systems 53

2.5.1 Conway Operators 54

2.5.2 Recursion in the Underlying Type System 55

2.5.3 Recursion in Refinement Type System 56

2.6 Related Work 56

2.7 Conclusion and Future Work 57

3 A Program Logic for Effect Handlers 59

3.1 Preliminaries 59

3.1.1 Ordered Object and Lax Slice Category 60

3.1.2 Monad Lifting 61

3.1.3 Algebraic Theory 62

3.1.4 Syntax and Semantics of Effect Handlers 63

3.2 Weakest Preconditions for Algebraic Operations 66
3.3 Operation-wise Condition for Lifting Algebras 68
3.4 Strong Monad Lifting 69
3.5 Operation-wise Condition for Lifting Simply Fibred Algebras . . 73
3.6 Program Logic 75
3.6.1 Generic Effects 76
3.6.2 With a Strong Monad Lifting 76
3.6.3 Without a Strong Monad Lifting 77
3.7 Related Work 78

4 Decision Tree-Based Ranking Function Synthesis 81
4.1 Preview by Examples 81
4.1.1 Termination Verification by CEGIS 81
4.1.2 Handling Cycles in Decision Tree Learning 82
4.2 (Non-)Termination Verification as Constraint Solving 85
4.3 CounterExample-Guided Inductive Synthesis (CEGIS) 87
4.4 Ranking Function Synthesis 89
4.4.1 Basic Definitions 89
4.4.2 Segmentation and (Explicit and Implicit) Cycles: One-Dimensional Case 90
4.4.3 Segmentation and (Explicit and Implicit) Cycles: Multi-Dimensional Lexicographic Case 91
4.4.4 Our Decision Tree Learning Algorithm 91
4.4.5 Improvement by Degenerating Negative Values 95
4.5 Implementation and Evaluation 95
4.6 Related Work 97
4.7 Conclusions and Future Work 98

5 Tail Probabilities of Randomized Programs via Higher Moments of Runtime 101
5.1 Preliminaries 101
5.2 Ranking Supermartingale for Higher Moments 103
5.2.1 Ranking Supermartingales for the Second Moments . . . 104
5.2.2 Ranking Supermartingales for the Higher Moments 105
5.3 From Moments to Tail Probabilities 106
5.4 Template-Based Synthesis Algorithm 107
5.4.1 Linear Template-Based Algorithm 107
5.4.2 Polynomial Template-based Algorithm 109
5.5 Experiments 111
5.6 Related Work 113
5.7 Conclusions and Future Work 116

A Full Definition of the Underlying Type System and the Dependent Refinement Type System 117
A.1 Underlying Type System 117
A.1.1 Typing Rules 117
A.1.2 Semantics 121
A.2 Refinement Type System 126
A.2.1 Typing Rules 126
A.2.2 Semantics 128

B Details of Chapter 5 **131**

B.1 Preliminaries on Measure Theory 131

B.2 Higher Moments of Runtimes and Rewards 132

B.3 Omitted Details and Proofs in Section 5.2 135

B.3.1 Basic Properties of the Pre-expectation 135

B.3.2 Basic Properties of the Time-Elapse Function 138

B.3.3 Characterizing Higher Moments as a Least Fixed Point . 138

B.3.4 Ranking Supermartingale for K-th Moments 142

B.4 Test Programs 143

B.5 Detailed Comparison with Existing Work 145

B.5.1 Comparison with [26] 145

B.5.2 Comparison with [59] 145

B.6 An Example of Polynomially Decreasing Tail Probability 146

Chapter 1

Introduction

This thesis is about the study of program semantics and its application to program verification. In this chapter, we describe the background of program semantics and program verification, introduce our approach, and outline this thesis.

1.1 Backgrounds

Semantics of programming languages. Semantics is a foundation of the study of programming languages. The meaning of a program is defined by interpreting the program in a mathematical model. A choice of a model determines what features of programming languages can be expressed by the model. Finding appropriate models is an essential topic of the study of program semantics.

Various program semantics have been introduced to capture various features of programming languages mathematically. One of the classical studies of program semantics is Scott's domain theory for untyped lambda calculus [104]. In domain theory, programs are interpreted as continuous functions in order-theoretic models, and notably, recursions are captured as least fixed points. Lambek applied categorical semantics to simply typed lambda calculus and showed that simply typed lambda calculus corresponds to cartesian closed categories [75]. Moggi used the category-theoretic notion of monads to uniformly capture computational effects such as input/output, exception, and nondeterminism [84]. Recently, categorical semantics have many applications, such as probabilistic programs [50] and effect handlers [92].

The semantics of programming languages not only defines the meaning of programs, but also allows us to discuss properties of programs. Hermida [49] studied a semantic framework of logical predicates/relations using the category-theoretic notion of fibrations. A fibration $p : \mathbb{E} \to \mathbb{B}$ has two layers of models. For example, we can use the base model $\mathbb{B} = \mathbf{Set}$ to interpret the simply typed lambda calculus using the cartesian closed structure of $\mathbb{B}$. As the other model $\mathbb{E}$, we can use the category of predicates on **Set** whose object is a set X together with a predicate on X. Then, $\mathbb{E}$ is also a cartesian closed category, and the interpretation of the simply typed lambda calculus in $\mathbb{E}$ gives the interpretation of logical predicates. The functor $p : \mathbb{E} \to \mathbb{B}$ maps the interpretation in $\mathbb{E}$ to the interpretation in $\mathbb{B}$, which can be understood as forgetting predicates. To sum up, we can show properties of a lambda term by considering a lifting of its interpretation along a fibration $p : \mathbb{E} \to \mathbb{B}$.

Program verification. Program verification is the problem of proving that a program satisfies a given specification, which describes desired behaviors of the program. There are various kinds of specifications to be verified. When we are interested in the program's input and output, we can specify the behavior like "for each input satisfying a (pre)condition, the output of the program satisfies another (post)condition". When we are interested in the runtime of the program, we can use specifications like "the program terminates within 100 steps". Computational effects can also be a target of specifications. We may consider a specification like "any possible output of the (nondeterministic) program satisfies some condition".

A naive way of proving such properties of programs is paper-and-pencil proofs. We can calculate the interpretation of a given program and write a proof of satisfaction of a specification as long as semantics is defined. However, paper-and-pencil proofs are error-prone and require substantial effort. To prevent errors in proofs, we can use interactive theorem provers such as Coq, Agda, and Isabelle. These theorem provers use expressive logics that can encode most of mathematics and allows us to write machine-checked proofs. However, interactive theorem provers still require substantial effort and expertise in the target program and interactive theorem provers themselves, which makes it difficult to apply this methodology to large-scale programs. Therefore, many studies aim at more convenient verification. We describe a few of them below.

Hoare logic [51] provides syntactic proof rules for Hoare triples. A Hoare triple $\{P\}c\{Q\}$ says that "if we run the program c from a state satisfying the precondition P, then the state after running c satisfies the postcondition Q". We can prove the validity of $\{P\}c\{Q\}$ by applying proof rules. Automation of Hoare logic is also studied. The weakest precondition transformer $\mathrm{wp}[c]$ [32] is one of the essential techniques to automate verification. Given a program c and a postcondition Q, we define the weakest precondition $\mathrm{wp}[c](Q)$ as the precondition such that $\{\mathrm{wp}[c](Q)\}c\{Q\}$ and for any P, if $\{P\}c\{Q\}$ holds, then $P \implies \mathrm{wp}[c](Q)$ holds. It follows that proving $\{P\}c\{Q\}$ is equivalent to calculating $\mathrm{wp}[c](Q)$ and proving $P \implies \mathrm{wp}[c](Q)$. The weakest precondition can be easily calculated except for while loops, for which we have to guess an invariant. Therefore, the weakest precondition transformer $\mathrm{wp}[c]$ enables us to extract verification conditions from the program c, and verification conditions can be checked by constraint solvers such as SMT (Satisfiability Modulo Theories) solvers and CHC (Constrained Horn Clauses) solvers.

Another kind of verification method is refinement type systems. Refinement type systems can describe specifications by refinement types $\{v : b \mid p\}$, which restricts values of type b by the predicate p. Verification conditions are extracted by applying typing rules. Specifically, typing rules for subtypings $\{v : b \mid p\} <: \{v : b \mid p\}$ between two refinement types yields the implication $p \implies q$ as a verification condition. Implementations of refinement type systems are also studied in [94, 110, 111, 124, 125].

1.2 Semantic Refinements

This thesis aims to extend the scope of program verification to new problems by what we call semantic refinements. The problem of program verification inevitably becomes more difficult if we consider programming language with more features. Making new verification methods on an ad hoc basis would not be manageable for complex problems. So, we pursue a more theoretically principled way of developing new verification methods. We utilize mathematical structures of models that are essential for verification methods to work well. We refine existing models so that refined models have more information about programs and maintain essential mathematical structures of existing models. With more information about programs, we can improve the expressivity and verification power of the existing methods. With the same mathematical structure, we can minimize modifications to syntactic inference rules or algorithms. Our semantic refinement includes Hermida's fibrational approach towards logical predicates/relations but is not necessarily limited to fibrations. We also study automated verification using semantic refinement as a part of this thesis.

This thesis contains four concrete topics. Each topic corresponds to one chapter.

1.2.1 A Categorical Construction of Dependent Refinement Type Systems

Dependent refinement types [39] 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 [14, 39, 67, 76, 116] and implemented in, e.g., F* [110, 111] and LiquidHaskell [94, 124, 125].

In this topic, 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.

$$\begin{tikzcd}
\{x \in \mathbb{Z} \mid x \geq 0\} \arrow[r, dashed] \arrow[d, phantom, sloped, "\subseteq"] & \{(x, v) \in \mathbb{Z} \times \mathbb{Z} \mid x \geq 0 \wedge v = x + 1\} \arrow[d, phantom, sloped, "\subseteq"] \\
\mathbb{Z} \arrow[r, "\langle \mathrm{id}_{\mathbb{Z}}, (-)+1 \rangle"] & \mathbb{Z} \times \mathbb{Z}
\end{tikzcd} \tag{1.1}$$

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. 1.1. 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 (1.1).

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

Figure 1.1: A lifting of a CCompC.

We extend the construction of liftings of CCompCs to liftings of fibred monads [8] 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 [116], and probability [14]. 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 [8].

We explain the usage of these categorical constructions by giving semantics to a dependent refinement type system with computational effects, which is based on [11]. 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 [11], 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 [9–11]) 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 [11]. We prove soundness of the typing rule for partial correctness of recursion under the existence of a lifting of Conway operators.

1.2.2 A Program Logic for Effect Handlers

Effect handlers and their verification. Effect handlers [92] provide a mechanism to implement user-defined computational effects. They are a generalization of exception handlers in two ways. Firstly, they can handle arbitrary algebraic operations (e.g. raising exceptions, input/output, nondeterministic choice, and read/write to the state of a program). Secondly, they can also specify how the rest of the computation (i.e. continuation) is resumed when handling an algebraic operation. These features allow us to flexibly define computational effects. Effect handlers are implemented in, e.g., Eff [16], Koka [77], and Multicore OCaml [34].

On the other hand, the flexibility of effect handlers makes it difficult to verify functional correctness of programs containing effect handlers. First of all, there are not many studies on the verification of effect handlers. Verifying functional correctness of effect handlers is studied in [81], but their method is ad hoc, and it was not known how we could apply their method to effect handlers for stateful computations. The problem we consider here is how we can provide a verification method for more general effect handlers.

Program logic for effect handlers. We provide a program logic for effect handlers in this chapter. We consider specifications in the style of the Hoare logic [40]. We use the following judgements.

$$\Gamma \mid K\ \{p\} \vdash M : \underline{C}\ \{q\} \qquad \Gamma \mid K\ \{p\} \vdash H : \underline{C}\ \{q\}\ \textbf{handler}$$

Here, we add a precondition p and a postcondition q to the typing judgements $\Gamma \mid K \vdash M : \underline{C}$ and $\Gamma \mid K \vdash M : \underline{C}$ **handler** for terms and effect handlers defined in [92]. We also provide inference rules for these judgements. Our inference rules are compositional in the sense that we can reduce verification of an effect handler to verification of each algebraic operation.

For example, consider the program in Fig. 1.2. We may consider a specification like "if $x > 0$ holds, then the program in Fig. 1.2 returns a value $\textbf{Some}(y)$ for some y". This specification can be expressed by the following judgement:

$$x : \textbf{int} \mid \diamond\ \{x > 0\} \vdash M : F(\textbf{option int})\ \{r.\exists y, r = \eta(\textbf{Some}(y))\}$$

$$\begin{array}{l}\textbf{if } x = 0 \textbf{ then } \mathrm{raise}(e_{\mathrm{ZeroDiv}}) \textbf{ else } 10/x \\ \quad \textbf{handled with } \{\mathrm{raise}(e; \_) \mapsto \textbf{return None}\} \\ \quad \textbf{to } y{:}\textbf{int in return Some}(y)\end{array}$$

Figure 1.2: An example of handling exception. If x is 0, then exception e_{ZeroDiv} is raised. The handler $\{\mathrm{raise}(e; \_) \mapsto$ **return None**$\}$ handles the exception and returns **None** of type **option int** := **None** | **Some int**. If x is not 0, then the program returns **Some**$(10/x)$.

$$\begin{tikzcd}
\mathbb{E} \arrow[d, "p"] & \llbracket \phi \rrbracket \arrow[r] & \llbracket \psi \rrbracket \\
\mathbb{B} & \llbracket \Gamma \rrbracket \arrow[r, "\llbracket M \rrbracket"] & \llbracket A \rrbracket
\end{tikzcd}$$

Figure 1.3: A lifting of an interpretation $\llbracket M \rrbracket$.

where M is the program in Fig. 1.2 and $\eta(\textbf{Some}(y))$ represent the pure computation that returns $\textbf{Some}(y)$.

Operation-wise liftings of Eilenberg-Moore algebras. The theoretical backgrounds of our program logic are 1) the relationship between fibrations and logical relations, which originate in [49], and 2) its application to Eilenberg-Moore algebras, which give semantics to effect handlers.

The relationship between fibrations and logical relations is studied in [49]. The idea is as follows (and shown in Fig. 1.3). Consider a category $\mathbb{B}$, in which a program is interpreted. Consider also a fibration $p : \mathbb{E} \to \mathbb{B}$ where the total category $\mathbb{E}$ consists of predicates on some object in $\mathbb{B}$, and the functor p maps a predicate on $X \in \mathbb{B}$ to the object X. Suppose that a precondition ϕ and a postcondition ψ are interpreted in the total category $\mathbb{E}$. In this situation, a lifting of the interpretation $\llbracket M \rrbracket$ of a program $\Gamma \vdash M : A$ along the fibration p represents a proof of correctness. Thus, rules for constructing a lifting of $\llbracket M \rrbracket$ can be understood as inference rules of a program logic.

The idea explained above can be extended for computational effects by considering *monad liftings*. Given a monad T on the base category $\mathbb{B}$, a lifting of T along the fibration $p : \mathbb{E} \to \mathbb{B}$ provides a mapping from a predicate on values in X to a predicate on effectful computations in TX. Monad liftings are studied in ,e.g., [8, 63, 64] and applied to study program logics in [6, 99]. However, none of these studies deal with effect handlers to the best of our knowledge. So, we extend these fibrational approach to a program logic for effect handlers in this thesis.

We consider liftings of the interpretation of an effect handlers to prove the correctness of the effect handler. Since the interpretation of an effect handler $H = \{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op}}$ is given by an Eilenberg-Moore algebra induced by the interpretation of M_{op} for each operation op, we aim to obtain liftings of the Eilenberg-Moore algebra from liftings of the interpretation of M_{op} for each algebraic operation. Such *operation-wise* liftings provide semantics of program logic that can verify effect handlers in an operation-wise manner (Fig. 1.4).

$$\frac{\text{correctness of } M_{\mathrm{op}} \text{ for each op} \in \Sigma}{\text{correctness of an effect handler } H} \qquad \frac{\text{a lifting of } [\![M_{\mathrm{op}}]\!] \text{ for each op} \in \Sigma}{\text{a lifting of } [\![\{\mathrm{op}(x;k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op}\in\Sigma}]\!]}$$

Figure 1.4: Operation-wise liftings of Eilenberg-Moore algebras provide semantics of our program logic for effect handlers.

When considering operation-wise liftings of Eilenberg-Moore algebras of a monad T, we use the domain fibration $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$ where the total category $\mathbf{Set}/\Omega$ is the lax slice category [8], which intuitively represents a category of Ω-valued predicates. There are two reasons for this restriction. We restrict the base category to **Set** because if we use **Set** and the monad T on **Set** has countable rank, then T has a corresponding algebraic theory. This algebraic theory gives a set of algebraic operations that can fully characterize the monad T. This is the first reason, and the same restriction is used in [92]. The other reason is as follows. We restrict the fibration to the domain fibration $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$ because if we use this fibration, then (a certain kind of) monad liftings of T correspond to monotone Eilenberg-Moore algebras on Ω [8]. This gives an algebraic characterization of monad liftings of T, which leads to useful conditions for operation-wise liftings of Eilenberg-Moore algebras. We believe that our restriction to the domain fibration $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$ is still useful and general enough to cover many examples of monads on **Set**. In particular, we deal with the state monad, which was out of the scope of the existing work [81].

Strong Monad Lifting. For the technical treatment of operation-wise liftings of Eilenberg-Moore algebras, we also need a lifting of the strength $\mathrm{st}_{X,Y} : X \times TY \to T(X \times Y)$ of a strong monad. When we consider the sequential composition of two programs $\Gamma \mid K \vdash M_1 : F\ A$ and $\Gamma, x : A \mid K \vdash M_2 : \underline{C}$, the lifting of the strength guarantees that the precondition on Γ of the first computation M_1 is reusable as a part of the precondition of the second computation M_2. We discuss conditions for the existence of a lifting of the strength and show that not every strong monad has a strong monad lifting (i.e. there exists the case where no lifting of $\mathrm{st}_{X,Y}$ exists for some X, Y). We also show that even if we do not have a strong monad lifting, we can guarantee the existence of a lifting of $\mathrm{st}_{X,Y}$ for some X, Y by restricting predicates. Thus, our program logic has two versions of inference rules: one is stronger but applicable only when we have a strong monad lifting, and the other is weaker but applicable even if we do not have a strong monad lifting.

Summary of contribution

- We provide sufficient conditions for operation-wise liftings of Eilenberg-Moore algebras along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$. This allows us to verify effect handlers in an operation-wise manner and leads to our program logic.
- We provide conditions for the existence of strong monad liftings along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$. We list several examples and non-examples of strong monad liftings.

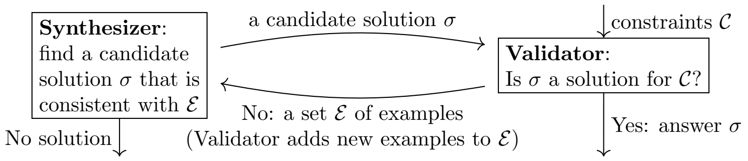


Figure 1.5: the CEGIS architecture

- We provide a compositional program logic for general effect handlers. Our framework can be applied to any combination of operations and equations as long as we have an appropriate choice of truth values Ω and a weakest precondition transformer for each operation.

1.2.3 Decision Tree-Based Ranking Function Synthesis

Termination Verification by Ranking Functions and CEGIS Termination verification is a fundamental but challenging problem in program analysis. Termination verification usually involves some well-foundedness arguments. Among them are those methods which synthesize *ranking functions* [40]: a ranking function assigns a natural number (or an ordinal, more generally) to each program state, in such a way that the assigned values strictly decrease along transition. Existence of such a ranking function witnesses termination, where well-foundedness of the set of natural numbers (or ordinals) is crucially used.

We study synthesis of ranking functions by CounterExample Guided Inductive Synthesis (CEGIS) [107]. CEGIS is an iterative learning model in which a synthesizer and a validator interact to find solutions for given constraints. At each iteration, (1) a synthesizer tries to find a candidate solution from the current examples, and (2) a validator accepts the candidate solution if it is correct, or rejects it providing counterexamples. These counterexamples are then used as part of the next examples (Fig. 1.5).

CEGIS has been applied not only to program verification tasks (synthesis of inductive invariants [43,44,87,88], that of ranking functions [46], etc.) but also to constraint solving (for CHC [25,36,98,127], for pwCSP($\mathcal{T}$) [117,118], etc.). The success of CEGIS is attributed to the degree of freedom that synthesizers enjoy. In CEGIS, synthesizers receive a set of individual examples that synthesizers can use in various creative and speculative manners (such as machine learning). In contrast, in other methods such as [12, 17–19, 78, 93], synthesizers receive logical constraints that are much more binding.

Segmented Synthesis in CEGIS-Based Termination Analysis The choice of a *candidate space* for candidate solutions σ is important in CEGIS. A candidate space should be *expressive*: by limiting a candidate space, the CEGIS architecture may miss a genuine solution. At the same time, *complexity* should be low: a larger candidate space tends to be more expensive for synthesizers to handle.

This tradeoff is also in the choice of the type of examples: using an expressive example type, a small number of examples can prune a large portion of the candidate space; however, finding such expressive examples tends to be expensive.

Table 1.1: ranking function synthesis by CEGIS

candidate space \ example type	trace examples	transition examples
affine ranking functions	[37,121]	[46]
piecewise affine ranking functions	[37,121]	our method

In this chapter, we use *piecewise affine functions* as our candidate space for ranking functions. Piecewise affine functions are functions of the form

$$f(\widetilde{x}) \quad = \quad \begin{cases} \widetilde{a}_1 \cdot \widetilde{x} + b_1 & \widetilde{x} \in L_1 \\ \vdots & \\ \widetilde{a}_n \cdot \widetilde{x} + b_n & \widetilde{x} \in L_n \end{cases} \tag{1.2}$$

where $\{L_1, \ldots, L_n\}$ is a partition of the domain of $f(\widetilde{x})$ such that each L_i is a polyhedron (i.e. a conjunction of linear inequalities). We say *segmented synthesis* to emphasize that our synthesis targets are piecewise affine functions with case distinction. Piecewise affine functions stand on a good balance between expressiveness and complexity: the tasks of synthesizers and validators can be reduced to linear programming (LP); at the same time, case distinction allows them to model a variety of situations, especially where there are discontinuities in the function values and/or derivatives.

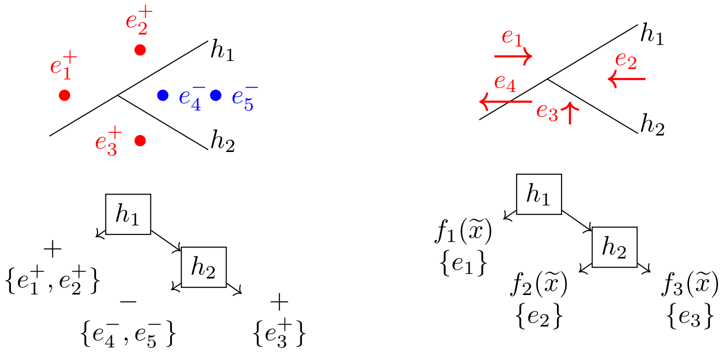
We use *transition examples* as our example type (Table 1.1). Transition examples are pairs of program states that represent transitions; they are much cheaper to handle compared to *trace examples* (finite traces of executions until termination) used e.g. in [37, 121]. The current work is the first to pursue segmented synthesis of ranking functions with transition examples; see Table 1.1.

Decision Tree Learning for CEGIS-Based Termination Analysis: a Challenge In this chapter, we represent piecewise affine functions (1.2) by the data structure of *decision trees*. The data structure suits the CEGIS architecture (Fig. 1.5): iterative refinement of candidate solutions can be naturally expressed by growing decision trees. The main challenge of this chapter is the design of an effective synthesizer for decision trees—such a synthesizer *learns* decision trees from examples.

In fact, decision tree learning in the CEGIS architecture has already been actively pursued, for the synthesis of *invariants* as opposed to ranking functions [25, 36, 44, 69, 127]. It is therefore a natural idea to adapt the decision tree learning algorithms used there, from invariants to ranking functions. However, we find that a naive adaptation of those algorithms for invariants does not suffice: they are good at handling *state examples* that appear in CEGIS for invariants; but they are not good at handling transition examples.

More specifically, when decision tree learning is applied to invariant synthesis (Fig. 1.6a), examples are given in the form of program states labeled as positive or negative. Decision trees are then built by iteratively selecting the best halfspaces—where "best" is in terms of some quality measures—until each leaf contains examples with the same label. One common quality measure used here is an information-theoretic notion of *information gain.*

We extend this from invariant synthesis to ranking function synthesis where examples are given by transitions instead of states (Fig. 1.6b). In this case,



(a) For invariants

(b) For ranking functions

Figure 1.6: Decision tree learning

a major challenge is to cope with examples that cross a border of the current segmentation—such as the transition e_4 crossing the border h_1 in Fig. 1.6b. Our decision tree learning algorithm should handle such crossing examples, taking into account the constraints imposed on the leaf labels affected by those examples (the affected leaf labels are $f_1(\widetilde{x})$ and $f_3(\widetilde{x})$ in the case of e_4).

Our Algorithm: Cycle-Based Decision Tree Learning for Transition Examples We use what we call the *cycle detection theorem* (Theorem 4.4.5) as a theoretical tool to handle such crossing examples. The theorem claims the following: if there is no piecewise affine ranking function with the current segmentation of the domain (such as the one in Fig. 1.6b given by h_1 and h_2), then this must be caused by a certain type of cycle of constraints, which we call an *implicit cycle*.

In our decision tree learning algorithm, when we do not find a piecewise affine ranking function with the current segmentation, we find an implicit cycle and refine the segmentation to break the cycle. Once all the implicit cycles are gone, the cycle detection theorem guarantees the existence of a candidate piecewise affine ranking function with the segmentation.

We integrate this decision tree learning algorithm in the CEGIS architecture (Fig. 1.5) and use it as a synthesizer. Our implementation of this framework gives promising experimental results on existing benchmark sets.

Contribution Our contribution is summarized as follows.

- We provide a decision tree-based synthesizer for ranking functions integrated into the CEGIS architecture. Our synthesizer uses transition examples to find candidate piecewise affine ranking functions. A major challenge here, namely handling constraints arising from crossing examples, is coped with by our theoretical observation of the cycle detection theorem.

- We implement our synthesizer for ranking functions implemented in MuVal and report the experience of using MuVal for termination and nontermination analysis. The experiment results show that MuVal's performance is comparable to state-of-the-art termination analyzers [18, 22, 35, 48] from Termination Competition 2020, and that MuVal can prove (non-)termination of some benchmarks with which other analyzers struggle.

1.2.4 Tail Probabilities of Randomized Programs via Higher Moments of Runtime

The important roles of *randomization* in algorithms and software systems are nowadays well-recognized. In algorithms, randomization can bring remarkable speed gain at the expense of small probabilities of imprecision. In cryptography, many encryption algorithms are randomized in order to conceal the identity of plaintexts. In software systems, randomization is widely utilized for the purpose of fairness, security and privacy.

Embracing randomization in programming languages has therefore been an active research topic for a long time. Doing so does not only offer a solid infrastructure that programmers and system designers can rely on, but also opens up the possibility of *language-based, static* analysis of properties of randomized algorithms and systems.

The current chapter's goal is to analyze imperative programs with randomization constructs—the latter come in two forms, namely probabilistic branching and assignment from a designated, possibly continuous, distribution. We shall refer to such programs as *randomized programs.*[1]

Runtime and Termination Analysis of Randomized Programs The *runtime* of a randomized program is often a problem of our interest; so is *almost-sure termination*, that is, whether the program terminates with probability 1. In the programming language community, these problems have been taken up by many researchers as a challenge of both practical importance and theoretical interest.

Most of the existing works on runtime and termination analysis follow either of the following two approaches.

- *Martingale-based methods*, initiated with a notion of *ranking supermartingale* in [23] and extended [5, 27, 28, 38, 54], have their origin in the theory of stochastic processes. They can also be seen as a probabilistic extension of *ranking functions*, a standard proof method for termination of (non-randomized) programs. Martingale-based methods have seen remarkable success in *automated synthesis* using templates and constraint solving (like LP or SDP).

- The *predicate-transformer* approach, pursued in [15, 59, 61], uses a more syntax-guided formalism of program logic and emphasizes reasoning by *invariants.*

The essential difference between the two approaches is not big: an invariant notion in the latter is easily seen to be an adaptation of a suitable notion of

[1] With the rise of statistical machine learning, *probabilistic programs* attract a lot of attention. Randomized programs can be thought of as a fragment of probabilistic programs without *conditioning* (or *observation*) constructs. In other words, the Bayesian aspect of probabilistic programs is absent in randomized programs.

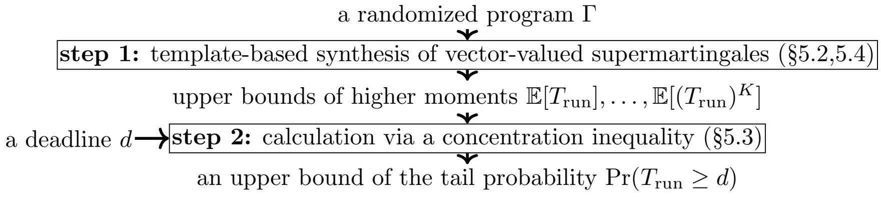

a randomized program Γ
step 1: template-based synthesis of vector-valued supermartingales (§5.2,5.4)
upper bounds of higher moments $\mathbb{E}[T_{\mathrm{run}}], \dots, \mathbb{E}[(T_{\mathrm{run}})^K]$
a deadline d
step 2: calculation via a concentration inequality (§5.3)
an upper bound of the tail probability $\Pr(T_{\mathrm{run}} \geq d)$

Figure 1.8: Our workflow

supermartingale. The work [112] presents a comprehensive account on the order-theoretic foundation behind these techniques.

These existing works are mostly focused on the following problems: deciding almost-sure termination, computing termination probabilities, and computing expected runtime. (Here "computing" includes giving upper/lower bounds.) See [112] for a comparison of some of the existing martingale-based methods.

Our Problem: Tail Probabilities for Runtimes In this chapter we focus on the problem of *tail probabilities* that is not studied much so far.[2] We present a method for *overapproximating* tail probabilities; here is the problem we solve.

Input: a randomized program Γ, and a *deadline* $d \in \mathbb{N}$

Output: an upper bound of the *tail probability* $\Pr(T_{\mathrm{run}} \geq d)$, where T_{run} is the runtime of Γ

Our target language is a imperative language that features randomization (probabilistic branching and random assignment). We also allow nondeterminism; this makes the program's runtime depend on the choice of a *scheduler* (i.e. how nondeterminism is resolved). In this chapter we study the longest, worst-case runtime (therefore our scheduler is *demonic*). In the technical sections, we use the presentation of these programs as *probabilistic control graphs (pCFGs)*—this is as usual in the literature. See e.g. [5, 112].

An example of our target program is in Fig. 1.7. It is an imperative program with randomization: in Line 3, the value of z is sampled from the uniform distribution over the interval $[-2, 1]$. The symbol in the line 4 stands for a nondeterministic Boolean value; in our analysis, it is resolved so that the runtime becomes the longest.

```
1  x := 2;  y := 2;
2  while (x > 0 && y > 0) do
3    z := Unif (-2,1);
4    if * then
5      x := x + z
6    else
7      y := y + z
8    fi
9  od
```

Figure 1.7: An example program

Given the program in Fig. 1.7 and a choice of a deadline (say $d = 400$), we can ask the question "what is the probability $\Pr(T_{\mathrm{run}} \geq d)$ for the runtime T_{run} of the program to exceed $d = 400$ steps?" As we show in Section 5.5, our method gives a guaranteed upper bound 0.0684. This means that, if we allow the time budget of $d = 400$ steps, the program terminates with the probability at least 93%.

Our Method: Concentration Inequalities, Higher Moments, and Vector-Valued Supermartingales Towards the goal of computing tail probabilities, our approach is to use *concentration inequalities*, a technique from

[2] An exception is [26]; see Section 5.6 for comparison with the current work.

probability theory that is commonly used for overapproximating various tail probabilities. There are various concentration inequalities in the literature, and each of them is applicable in a different setting, such as a nonnegative random variable (Markov's inequality), known mean and variance (Chebyshev's inequality), a difference-bounded martingale (Azuma's inequality), and so on. Some of them were used for analyzing randomized programs [26] (see Section 5.6 for comparison).

In this chapter, we use a specific concentration inequality that uses *higher moments* $\mathbb{E}[T_{\mathrm{run}}], \ldots, \mathbb{E}[(T_{\mathrm{run}})^K]$ of runtimes T_{run}, up to a choice of the maximum degree K. The concentration inequality is taken from [21]; it generalizes Markov's and Chebyshev's. We observe that a higher moment yields a tighter bound of the tail probability, as the deadline d grows bigger. Therefore it makes sense to strive for computing higher moments.

For computing higher moments of runtimes, we systematically extend the existing theory of ranking supermartingales, from the expected runtime (i.e. the first moment) to higher moments. The theory features a *vector-valued* supermartingale, which not only generalizes easily to degrees up to arbitrary $K \in \mathbb{N}$, but also allows automated synthesis much like usual supermartingales.

We also claim that the soundness of these vector-valued supermartingales is proved in a mathematically clean manner. Following [112], our arguments are based on the order-theoretic foundation of fixed points (namely the Knaster-Tarski, Cousot–Cousot and Kleene theorems), and we give upper bounds of higher moments by suitable least fixed points.

Overall, our workflow is as shown in Fig. 1.8. We note that the step 2 in Fig. 1.8 is computationally much cheaper than the step 1: in fact, the step 2 yields a symbolic expression for an upper bound in which d is a free variable. This makes it possible to draw graphs like the ones in Fig. 5.2. It is also easy to find a deadline d for which $\Pr(T_{\mathrm{run}} \geq d)$ is below a given threshold $p \in [0, 1]$.

We implemented a prototype that synthesizes vector-valued supermartingales using linear and polynomial templates. The resulting constraints are solved by LP and SDP solvers, respectively. Experiments show that our method can produce nontrivial upper bounds in reasonable computation time. We also experimentally confirm that higher moments are useful in producing tighter bounds.

Our Contributions Summarizing, the contribution of this chapter is as follows.

- We extend the existing theory of ranking supermartingales from expected runtimes (i.e. the first moment) to *higher moments*. The extension has a solid foundation of order-theoretic fixed points. Moreover, its clean presentation by vector-valued supermartingales makes automated synthesis as easy as before. Our target randomized programs are rich, embracing nondeterminism and continuous distributions.

- We study how these vector-valued supermartingales (and the resulting upper bounds of higher moments) can be used to yield upper bounds of *tail probabilities of runtimes*. We identify a concentration lemma that suits this purpose. We show that higher moments indeed yield tighter bounds.

- Overall, we present a comprehensive language-based framework for overapproximating tail probabilities of runtimes of randomized programs (Fig. 1.8). It has been implemented, and our experiments suggest its practical use.

1.3 Outline of This Thesis and Corresponding Papers

We study refinements in categorical semantics in Chapter 2,3 and extension of existing algorithm via refinements in Chapter 4,5.

In Chapter 2, we study a general construction of a categorical model of dependent refinement type systems. In Section 2.1, we review preliminaries on fibrations and closed comprehension categories, which give categorical models of dependent type systems. In Section 2.2 and Section 2.3, we provide a general construction of a categorical model of dependent refinement type systems together with the proof that it also has the structure of closed comprehension categories. In Section 2.4, we explain how the constructed model can interpret dependent refinement type systems by defining the interpretation concretely. In Section 2.5, we consider Conway operators for interpreting recursions in dependent refinement type systems. We discuss related work in Section 2.6 and conclude in Section 2.7. Chapter 2 is based on our paper [70] and extended with detailed proofs.

In Chapter 3, we study a program logic for effect handlers. In Section 3.1, we explain preliminaries on monad liftings and the syntax and semantics of effect handlers. In Section 3.2, we show that choosing (a certain kind of) a monad lifting is equivalent to specifying a weakest precondition transformer for each algebraic operation. In Section 3.3, we provide a necessary and sufficient condition for operation-wise liftings of Eilenberg-Moore algebras in a simplified situation. This condition is expressed using the weakest precondition transformer for each algebraic operation. To support operation-wise liftings in a more general situation, we consider liftings of the strength of a strong monad in Section 3.4 In Section 3.5, we provide the main theoretical results on sufficient conditions for operation-wise liftings, which allows us to construct a lifting of the interpretation of an effect handlers $H = \{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op}}$ from a lifting of the interpretation of M_{op} for each algebraic operation op. In Section 3.6, we provide a compositional program logic for effect handlers. We discuss related work in Section 3.7.

In Chapter 4, we study an example-based termination analyzer. Section 4.1 shows the overview of our method via examples. Section 4.2 explains our target class of predicate constraint satisfaction problems and how to encode (non-)termination problem into such constraints. In Section 4.3, we review CEGIS architecture, and then explain simplification of examples into positive/negative examples. Section 4.4 proposes our main contribution, our decision tree-based ranking function synthesizer. Section 4.5 shows our implementation and experimental results. Related work is discussed in Section 4.6, and we conclude in Section 4.7. Chapter 4 is based on [71].

In Chapter 5, we study a method to give an upper bound of a tail probability of a randomized program. We give preliminaries in Section 5.1. In Section 5.2, we review the order-theoretic characterization of ordinary ranking supermartingales and present an extension to higher moments of runtimes. In Section 5.3, we discuss how to obtain an upper bound of the tail probability of runtimes. In Section 5.4, we explain an automated synthesis algorithm for our ranking supermartingales. In Section 5.5, we give experimental results. In Section 5.6, we discuss related work. We conclude and give future work in Section 5.7. Chapter 5 is based on [72].

Chapter 2

A Categorical Construction of Dependent Refinement Type Systems

In this chapter, we provide a general construction of a categorical model of a dependent refinement type system from a model of an underlying type system and a model of predicate logic.

2.1 Preliminaries

We review basic notions and define their notations.

For basic category theoretic notions, we use the following notations. The identity morphism is denoted by $\mathrm{id}_X : X \to X$, and the composite of $f : X \to Y$ and $g : Y \to Z$ is denoted by $g \circ f : X \to Z$. The identity functor is denoted by $\mathrm{Id}_{\mathbb{C}} : \mathbb{C} \to \mathbb{C}$ and the composite of two functors is denoted by juxtaposition. We denote a terminal object by 1 and the unique morphism to 1 by $! : X \to 1$. The tupling of $f : X \to Y$ and $g : X \to Z$ is denoted by $\langle f, g \rangle : X \to Y \times Z$. The projections are denoted by $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$. The cotupling of $f : X \to Z$ and $g : Y \to Z$ is denoted by $[f, g] : X + Y \to Z$. The coprojections are denoted by $\iota_1 : X \to X+Y$ and $\iota_2 : Y \to X+Y$. The unit and the counit of an adjunction $F \dashv U$ is denoted by $\eta^{F \dashv U}$ and $\epsilon^{F \dashv U}$, respectively, but we often omit superscript if it is clear from the context. Exponentials are denoted by $X \Rightarrow Y$ or Y^X.

2.1.1 Fibration

We review basic concepts on fibrations (see e.g. [53] for details).

Let $p : \mathbb{E} \to \mathbb{B}$ be a functor. For any $u : I \to J$ in $\mathbb{B}$, we say a morphism $f : X \to Y$ in $\mathbb{E}$ is over u if $pf = u$ holds. A *vertical morphism* is a morphism over an identity morphism. Given a morphism $u : I \to J$, a *Cartesian morphism* (or *p-Cartesian morphism*) over u (or *Cartesian lifting* of u) is a morphism $f : X \to Y$ such that $pf = u$ holds and f satisfies the *Cartesian lifting property*: for each $v : K \to I$ and a morphism $h : Z \to Y$ over $u \circ v$, we have a unique morphism $g : Z \to X$ over v such that $h = f \circ g$. The functor $p : \mathbb{E} \to \mathbb{B}$ is a *fibration* if for each $Y \in \mathbb{E}$ and a morphism $u : I \to pY$ in $\mathbb{B}$, there exists a Cartesian morphism $\overline{u}(Y) : u^*Y \to Y$ over u.

A *fibre category* $\mathbb{E}_I$ over $I \in \mathbb{B}$ is a category defined by $\mathbb{E}_I = p^{-1}I$, that is, an object is $X \in \mathbb{E}$ such that $pX = I$ and a morphism $f : X \to Y$ is a vertical morphism in $\mathbb{E}$. Given a choice of a Cartesian lifting $\overline{u}(Y) : u^*Y \to Y$ for each u and Y, we have a *reindexing functor* $u^* : \mathbb{E}_J \to \mathbb{E}_I$ for each $u : I \to J$, and canonical natural isomorphisms $u^*v^* \cong (v \circ u)^*$ and $\mathrm{id}_I^* \cong \mathrm{Id}_{\mathbb{E}_I}$ satisfying some coherent conditions. A *split fibration* is a fibration together with a choice of Cartesian morphisms such that these canonical natural isomorphisms become identities.

Opfibrations are a dual notion of fibrations: a functor $p : \mathbb{E} \to \mathbb{B}$ is an *opfibration* if for each $X \in \mathbb{E}$ and $u : pX \to J$, we have a coCartesian morphism $f : X \to \underline{u}(X)$ over u. Here, a *coCartesian morphism* over $u : I \to J$ is a morphism $f : X \to Y$ over u such that for each $v : J \to K$ and a morphism $h : X \to Z$ over $v \circ u$, there exists a unique morphism $g : Y \to Z$ over v such that $h = g \circ f$ holds.

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.

Example 2.1.1 (subobject fibration [53, §1.3])**.** Let $\mathbf{Sub}(\mathbf{Set})$ be the category of subobjects in $\mathbf{Set}$: an object is a pair $(I, X \subseteq I)$ where $I \in \mathbf{Set}$, and a morphism $f : (I, X) \to (J, Y)$ is a morphism $f : I \to J$ in $\mathbf{Set}$ such that there exists a (unique) morphism $f' : X \to Y$ that is a restriction of f (in other words, for each $x \in X$, we have $f(x) \in Y$). We define a functor $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ by $\mathsf{sub}_{\mathbf{Set}}(I, X) = I$ and $\mathsf{sub}_{\mathbf{Set}} f = f$. Then, $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ is a posetal fibration.

Lemma 2.1.2. *Let $p : \mathbb{E} \to \mathbb{B}$ be a posetal fibration, and $f : X \to Y$ be a morphism in $\mathbb{E}$ over an isomorphism $u : I \to J$. The following conditions are equivalent.*

1. *f is isomorphic.*

2. *f is Cartesian.*

3. *$X \geq u^*Y$.*

Proof. (2 iff 3) In posetal fibration, f is Cartesian if and only if $X = u^*Y$. Since $X \leq u^*Y$ follows from the existence of f, f is Cartesian if and only if $X \geq u^*Y$.

(1 implies 3) If f is isomorphic, then f^{-1} must be over u^{-1}. We have $X \leq u^*Y$ and $Y \leq (u^{-1})^*X$, which imply $u^*Y \leq u^*(u^{-1})^*X = X$.

(3 implies 1) If $X \geq u^*Y$, then there exists a morphism $g : Y \to X$ defined by $Y = (u^{-1})^*u^*Y \leq (u^{-1})^*X \xrightarrow{\overline{u^{-1}}(X)} X$. Then, g is the inverse of f because q is faithful, and $f \circ g$ and $g \circ f$ are over $u \circ u^{-1} = \mathrm{id}_J$ and $u^{-1} \circ u = \mathrm{id}_I$, respectively. □

We often use the word "lifting" along a fibration in this thesis. The formal definition is given on a case-by-case basis, but the common intuition can be understood as follows. Let $\diamondsuit$ be a category theoretic notion (e.g. monad) and $p : \mathbb{E} \to \mathbb{B}$ be a fibration. Roughly speaking, given an instance X of $\diamondsuit$ in the base category $\mathbb{B}$, a lifting of X is an instance Y of $\diamondsuit$ in the total category $\mathbb{E}$ such that p maps the structure of Y to that of X.

2.1.2 Comprehension Category

We review basic definitions and fix notations for comprehension categories, which are used as categorical models for dependent type theories.

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. Note that the notation π_X has the same form as the projections π_1, π_2 for binary products, but they can be distinguished by the subscript.

The comprehension category with unit $p : \mathbb{E} \to \mathbb{B}$ induces several structures. It induces a comprehension category $\mathcal{P} : \mathbb{E} \to \mathbb{B}^{\to}$ 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 $\phi : \mathbb{E}_{\{X\}}(1\{X\}, \pi_X^* Y) \cong \mathbb{E}_I(X, Y)$. We have the pullback square $\mathcal{P}(\overline{\pi_X}(Y))$ for each $X, Y \in \mathbb{E}_I$, and this induces 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 $\{\overline{\pi_X}(Y)\} = \pi_{\pi_Y^* X} \circ \sigma_{X,Y}$ by the universal property of pullbacks. 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 [53, 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 $\{\overline{\pi_X}(\coprod_X Y) \circ \eta_Y^{\pi_X^* \dashv \coprod_X}\}$ is an isomorphism for each $X \in \mathbb{E}$ and $Y \in \mathbb{E}_{\{X\}}$. Intuitively, products $\prod_X$ and strong coproducts $\coprod_X$ in CCompCs represent dependent function types $(\Gamma, x : A \vdash B) \mapsto (\Gamma \vdash \Pi x{:}A.B)$ and dependent pair types $(\Gamma, x : A \vdash B) \mapsto (\Gamma \vdash \Sigma x{:}A.B)$, respectively. 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).

Example 2.1.3 (simple fibration)**.** Let $\mathbb{B}$ be a category with finite products. We define a category $\mathbf{s}(\mathbb{B})$ as follows. An object $(I, X) \in \mathbf{s}(\mathbb{B})$ is a pair of objects in $\mathbb{B}$. A morphism $(u, f) : (I, X) \to (J, Y)$ is a pair of morphisms $u : I \to J$ and $f : I \times X \to Y$. Identity morphisms are given by $(\mathrm{id}_I, \pi_2) : (I, X) \to (I, X)$, and the composition is given by $(v, g) \circ (u, f) \coloneqq (v \circ u, g \circ \langle u \circ \pi_1, f\rangle)$.

The *simple fibration* $\mathsf{s}_{\mathbb{B}} : \mathbf{s}(\mathbb{B}) \to \mathbb{B}$ is defined by $\mathsf{s}_{\mathbb{B}}(I, X) \coloneqq I$ and $\mathsf{s}_{\mathbb{B}}(u, f) \coloneqq u$ [53, Definition 1.3.1]. The simple fibration is a SCCompC if and only if $\mathbb{B}$ is cartesian closed [53, Theorem 10.5.5]. Specifically, the terminal object functor is given by $I \mapsto (I, 1)$, the comprehension functor by $\{(I, X)\} = I \times X$, products by $\prod_{(I,X)}(I \times X, Y) = (I, X \Rightarrow Y)$, and coproducts by $\coprod_{(I,X)}(I \times X, Y) = (I, X \times Y)$.

Example 2.1.4 (family fibration)**.** We define a category $\mathbf{Fam}(\mathbf{Set})$ as follows. An object $(I, X) \in \mathbf{Fam}(\mathbf{Set})$ is a pair of a set I and a family $X : I \to \mathbf{Set}$ of sets indexed by I. A morphism $(u, f) : (I, X) \to (J, Y)$ is a pair of a function $u : I \to J$ and a family of functions $f_i : X_i \to Y_{u(i)}$ for each $i \in I$. Identity morphisms are given by $(\mathrm{id}_I, \{\mathrm{id}_{X_i}\}_{i \in I}) : (I, X) \to (I, X)$, and the composition is given by $(v, g) \circ (u, f) \coloneqq (v \circ u, \{g_{u(i)} \circ f_i\}_{i \in I})$ where $(v, g) : (J, Y) \to (K, Z)$ and $(u, f) : (I, X) \to (J, Y)$.

The family fibration $\mathsf{fam}_{\mathbf{Set}} : \mathbf{Fam}(\mathbf{Set}) \to \mathbf{Set}$ is defined by $\mathsf{fam}_{\mathbf{Set}}(I, X) \coloneqq I$ and $\mathsf{fam}_{\mathbf{Set}}(u, f) \coloneqq u$. This is a SCCompC: the terminal object functor is given by $I \mapsto (I, \{1\}_{i \in I})$, the comprehension functor by the disjoint union $\{(I, X)\} = \coprod_{i \in I} X_i$, products by a family of sets of dependent functions

$$\prod_{(I,X)} (\{(I, X)\}, Y) = (I, \{\prod_{x \in X_i} Y_{(i,x)}\}_{i \in I}),$$

coproducts by a family of sets of dependent pairs

$$\coprod_{(I,X)} (\{(I, X)\}, Y) = (I, \{\coprod_{x \in X_i} Y_{(i,x)}\}_{i \in I}).$$

When $p : \mathbb{E} \to \mathbb{B}$ is a CCompC, we define $\mathbf{fst} \in \mathbb{E}_I(\coprod_X Y, X)$ as a unique morphism that is mapped by $\mathcal{P} : \mathbb{E} \to \mathbb{B}^{\to}$ to the following morphism.

$$\left(\mathbf{fst} : \coprod_X Y \to X\right) \quad \overset{\mathcal{P}}{\longmapsto} \quad \left(\begin{tikzcd} \{\coprod_X Y\} \arrow[r, "\kappa^{-1}"'] \arrow[d, "\pi_{\coprod_X Y}"] & \{Y\} \arrow[r, "\pi_Y"'] & \{X\} \arrow[d, "\pi_X"] \\ I \arrow[rr, equal] & & I \end{tikzcd}\right)$$

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 $A + B$.

Lemma 2.1.5. The symmetry isomorphism $\sigma_{X,Y} : \{\pi_X^* Y\} \to \{\pi_Y^* X\}$ satisfies the following properties.

- $\pi_{\pi_Y^* X} \circ \sigma_{X,Y} = \{\overline{\pi_X}(Y)\}$ and $\{\overline{\pi_Y}(X)\} \circ \sigma_{X,Y} = \pi_{\pi_X^* Y}$
- $\sigma_{Y,X} \circ \sigma_{X,Y} = \mathrm{id}_{\{\pi_X^* Y\}}$

- $\{\pi_Y^* f\} \circ \sigma_{X',Y} = \sigma_{X,Y} \circ \{\overline{\{f\}}(\pi_X^* Y)\}$ for each $f \in \mathbb{E}_I(X', X)$
- $\{f'\} \circ \sigma_{X',(pf)^*Y} = \sigma_{X,Y} \circ \{\overline{\{f\}}(\pi_X^* Y)\}$ for $X, Y \in \mathbb{E}_I$, $X' \in \mathbb{E}_{I'}$ and $f : X' \to X$ where f' is defined by the unique morphism in the following diagram.

$$\begin{tikzcd}
\pi^*_{(pf)^*Y} X' \arrow[r, "\overline{\pi}(X')"] \arrow[d, dashed, "f'"'] & X' \arrow[d, "f"] \\
\pi^*_Y X \arrow[r, "\overline{\pi}(X)"] & X
\end{tikzcd}$$

$$\begin{tikzcd}
\{(pf)^*Y\} \arrow[r, "\pi"] \arrow[d, "\{\overline{pf}(Y)\}"'] & I' \arrow[d, "pf"] \\
\{Y\} \arrow[r, "\pi"] & I
\end{tikzcd}$$

Proof. The first and the second properties are obvious from the definition. The third property is a special case of the forth.

The forth property follows from universal property of pullbacks in the following diagram.

$$\begin{tikzcd}
\{\pi^*_{X'}(pf)^*Y\} \arrow[r, equal] \arrow[rrr, bend left, "\{\overline{\pi_{X'}}((pf)^*Y)\}"] \arrow[d, "\sigma_{X',(pf)^*Y}"] \arrow[dd, bend right, "\pi_{\dots}"'] & \{\{f\}^*\pi^*_X Y\} \arrow[r, "\{\overline{\{f\}}(\pi^*_X Y)\}"'] & \{\pi^*_X Y\} \arrow[d, "\sigma_{X,Y}"'] \arrow[dr, "\{\overline{\pi_X}(Y)\}"] & \{(pf)^*Y\} \arrow[d, "\{\overline{pf}(Y)\}"'] \arrow[ddd, bend left, "\pi_{\dots}"] \\
\{\pi^*_{(pf)^*Y} X'\} \arrow[rr, "\{f'\}"] \arrow[d, "\{\overline{\pi}(X)\}"] & & \{\pi^*_Y X\} \arrow[r, "\pi_{\pi^*_Y X}"'] \arrow[d, "\{\overline{\pi_Y}(X)\}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \{Y\} \arrow[d, "\pi_Y"] \\
\{X'\} \arrow[rr, "\{f\}"] \arrow[rrrd, "\pi_{X'}"'] & & \{X\} \arrow[r, "\pi_X"] & I \\
& & & I' \arrow[u, "pf"]
\end{tikzcd}$$

□

2.2 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.

2.2.1 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 2.2.1. 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 $p : \mathbb{E} \to \mathbb{B}$.

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

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 $\{f\} = qh$.

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

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. Note also that two morphisms $(f_1, g_1, h_1), (f_2, g_2, h_2) : (X, P, Q) \to (X', P', Q')$ in $\{\mathbb{E} \mid \mathbb{P}\}$ are equal if and only if f_1 and f_2 are equal because q is faithful. This is often used to show properties of $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$.

The functor $\{p \mid q\}$ inherits most of the CCompC structure of $p : \mathbb{E} \to \mathbb{B}$ as we will see below.

The functor $\{p \mid q\}$ is a split fibration.

Lemma 2.2.2. A morphism $(f, g, h) : (X, P, Q) \to (X', P', Q')$ is $\{p \mid q\}$-Cartesian if and only if $f : X \to X'$ is p-Cartesian and $Q = \pi_X^* P \wedge \{f\}^* Q'$.

Proof. For the forward direction, assume that $(f, g, h) : (X, P, Q) \to (X', P', Q')$ is $\{p \mid q\}$-Cartesian.

We first prove that f is p-Cartesian. Let $u = pf$, and $f'' : X'' \to X'$ be a morphism over $u \circ u'$ where $u' : pX'' \to pX$. There exists a morphism $f' : X'' \to X$ over u' such that $f'' = f \circ f'$ because $(f, g, h) : (X, P, Q) \to (X', P', Q')$ in the diagram below is $\{p \mid q\}$-Cartesian.

$$\begin{tikzcd}
 & (X'', (u')^*P, \{f''\}^*Q' \wedge \pi_{X''}^*(u')^*P) \arrow[d, dashed, "{(f', \overline{u'}(P), h')}"'] \arrow[dr, "{(f'', g \circ \overline{u'}(P), \overline{\{f''\}}(Q') \circ \pi_1)}"] & \\
\{\mathbb{E} \mid \mathbb{P}\} \arrow[d, "p"] & (X, P, Q) \arrow[r, "{(f,g,h)}"'] & (X', P', Q') \\
\mathbb{P} & (u')^*P \arrow[d, "\overline{u'}(P)"'] \arrow[dr] & \\
 & P \arrow[r, "g"'] & P'
\end{tikzcd}$$

Uniqueness of such $f' : X'' \to X$ is shown as follows. If there are two morphisms $f_1', f_2' : X'' \to X$ over u' such that $f'' = f \circ f_1' = f \circ f_1'$, then $f_1' = f_2'$ follows from the $\{p \mid q\}$-Cartesianness of (f, g, h) in the following diagram.

$$\begin{tikzcd}
 & (X'', (u')^*P, \{f_1'\}^*Q \wedge \{f_2'\}^*Q \wedge \pi_{X''}^*(u')^*P) \arrow[d, shift right, "{(f_1',\overline{u'}(P),\overline{\{f_1'\}}(Q)\circ\pi_1)}"'] \arrow[d, shift left, "{(f_2',\overline{u'}(P),\overline{\{f_2'\}}(Q)\circ\pi_2)}"] \arrow[dr, "{(f'',g\circ\overline{u'}(P),\dots)}"] & \\
\{\mathbb{E} \mid \mathbb{P}\} \arrow[d, "\{p|q\}"] & (X, P, Q) \arrow[r, "{(f,g,h)}"'] & (X', P', Q') \\
\mathbb{P} & & \\
 & (u')^*P \arrow[d, "\overline{u'}(P)"'] \arrow[dr] & \\
 & P \arrow[r, "g"'] & P'
\end{tikzcd}$$

Therefore, f is p-Cartesian.

Next, we prove $Q = \pi_X^*P \wedge \{f\}^*Q'$. By definition of $\{\mathbb{E} \mid \mathbb{P}\}$, we have $Q \leq \pi_X^*P$ and $Q \leq \{f\}^*Q'$, so it suffices to prove $Q \geq \pi_X^*P \wedge \{f\}^*Q'$. Consider the following diagram.

$$\begin{tikzcd}
 & (X, P, \pi_X^*P \wedge \{f\}^*Q') \arrow[d, dashed, "{(f',\mathrm{id}_P,h')}"'] \arrow[dr, "{(f,g,\overline{\{f\}}(Q')\circ\pi_2)}"] & \\
\{\mathbb{E} \mid \mathbb{P}\} \arrow[d, "\{p|q\}"] & (X, P, Q) \arrow[r, "{(f,g,h)}"'] & (X', P', Q') \\
\mathbb{P} & & \\
 & P \arrow[d, equal] \arrow[dr] & \\
 & P \arrow[r, "g"'] & P'
\end{tikzcd}$$

There exists a unique morphism (f', id_P, h'). Since f is p-Cartesian, we have $f' = \mathrm{id}_X$ and thus $Q \geq \pi_X^*P \wedge \{f\}^*Q'$.

For the converse direction, consider a morphism $(f, g, h) : (X, P, Q) \to (X', P', Q')$ such that $f : X \to X'$ is p-Cartesian and $Q = \pi_X^*P \wedge \{f\}^*Q'$. Suppose that we have a morphism $(f'', g \circ g', h'') : (X'', P'', Q'') \to (X', P', Q')$ over $g \circ g'$ where $g' : P'' \to P$. Since $p : \mathbb{E} \to \mathbb{B}$ is a fibration, we have a unique $f' : X'' \to X$ such that $pf' = qg'$ and $f'' = f \circ f'$. We also have $h' : Q'' \to Q$ over $\{f'\}$ because the following inequality holds.

$$\begin{aligned}
\{f'\}^*Q &= \{f'\}^*(\pi_X^*P \wedge \{f\}^*Q') \\
&= \pi_{X''}^*(pf')^*P \wedge \{f''\}^*Q' \\
&\geq \pi_{X''}^*P'' \wedge Q'' \\
&= Q''
\end{aligned}$$

By the uniqueness of f' and the faithfulness of $q : \mathbb{P} \to \mathbb{B}$, it is easy to check that $(f', g', h') : (X'', P'', Q'') \to (X, P, Q)$ is a unique morphism over g' such that the following diagram commutes.

$$\begin{tikzcd}
(X'', P'', Q'') \arrow[d, dashed, "{(f',g',h')}"'] \arrow[dr, "{(f'',g\circ g',h'')}"] & \\
(X, P, Q) \arrow[r, "{(f,g,h)}"] & (X', P', Q')
\end{tikzcd}$$

Therefore, $(f, g, h) : (X, P, Q) \to (X', P', Q')$ is $\{p \mid q\}$-Cartesian. □

Lemma 2.2.3. The functor $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ is a split fibration. For each $g : P' \to P$ in $\mathbb{P}$ and $(X, P, Q) \in \{\mathbb{E} \mid \mathbb{P}\}$, the $\{p \mid q\}$-Cartesian morphism $\overline{g}(X, P, Q) : g^*(X, P, Q) \to (X, P, Q)$ over g is given as follows.

$$\begin{aligned} g^*(X, P, Q) &\coloneqq ((qg)^*X, P', \pi^*_{(qg)^*X}P' \wedge \{\overline{qg}(X)\}^*Q) \\ \overline{g}(X, P, Q) &\coloneqq (\overline{qg}(X), g, \overline{\{\overline{qg}(X)\}}(Q) \circ \pi') \end{aligned}$$

where $\pi' : \pi^*_{(qg)^*X}P' \wedge \{\overline{qg}(X)\}^*Q \to \{\overline{qg}(X)\}^*Q$ is the projection.

Proof. The morphism $\overline{g}(X, P, Q)$ is $\{p \mid q\}$-Cartesian by Lemma 2.2.2. The fibration $\{p \mid q\}$ is split because p is split. □

The functor $\{p \mid q\}$ has a terminal object functor.

Definition 2.2.4. We define a functor $1 : \mathbb{P} \to \{\mathbb{E} \mid \mathbb{P}\}$ by $1P \coloneqq (1qP, P, \pi^*_{1qP}P)$ and $1g \coloneqq (1qg, g, h)$ for each $P \in \mathbb{P}$ and $g : P \to P'$ where h is a unique morphism in the diagram below.

$$\begin{tikzcd}
 & \pi^*_{1qP}P \arrow[r, "\overline{\pi_{1qP}}(P)"] \arrow[d, dashed, "h"] & P \arrow[d, "g"] \\
\mathbb{P} \arrow[dd, "q"] & \pi^*_{1qP'}P' \arrow[r, "\overline{\pi_{1qP'}}(P')"] & P' \\
 & & \\
\mathbb{B} & \{1qP\} \arrow[r, "\pi_{1qP}"] \arrow[d, "\{1qg\}"] & qP \arrow[d, "qg"] \\
 & \{1qP'\} \arrow[r, "\pi_{1qQ'}"] & qP'
\end{tikzcd}$$

Lemma 2.2.5. The functor $1 : \mathbb{P} \to \{\mathbb{E} \mid \mathbb{P}\}$ is a fibred terminal object.

Proof. We prove that we have a fibred adjunction $\{p \mid q\} \dashv 1 : \mathrm{Id}_{\mathbb{P}} \to \{p \mid q\}$ in the 2-category $\mathbf{Fib}_{\mathbb{P}}$.

First, we prove that $1 : \mathbb{P} \to \{\mathbb{E} \mid \mathbb{P}\}$ is a fibred functor, i.e., preserves Cartesian morphisms. It suffices to prove $1g = (1qg, g, h) : (1qP, P, \pi^*_{1qP}P) \to (1qP', P', \pi^*_{1qP'}P')$ is $\{p \mid q\}$-Cartesian for each $g : P \to P'$ in $\mathbb{P}$ because every morphism in $\mathbb{P}$ is $\mathrm{Id}_{\mathbb{P}}$-Cartesian. We use Lemma 2.2.2. The first component $1qg : 1qP \to 1qP'$ is p-Cartesian because the fibred functor $1 : \mathrm{Id}_{\mathbb{B}} \to p$ preserves Cartesian morphisms. We also have

$$\begin{aligned} \pi^*_{1qP}P \wedge \{1qg\}^*(\pi^*_{1qP'}P') &= \pi^*_{1qP}P \wedge \pi^*_{1qP}(qg)^*P' \\ &= \pi^*_{1qP}(P \wedge (qg)^*P') \\ &= \pi^*_{1qP}P. \end{aligned}$$

Next, We define vertical natural transformations $\eta : \mathrm{Id}_{\{\mathbb{E}|\mathbb{P}\}} \to 1\{p \mid q\}$ and $\epsilon : \{p \mid q\}1 \to \mathrm{Id}_{\mathbb{P}}$ by

$$\begin{aligned} \eta_{(X,P,Q)} &= (\eta^{p \dashv 1}_X, \mathrm{id}_P, h) &&: \quad (X, P, Q) \to (1qP, P, \pi^*_{1qP}P) \\ \epsilon_P &= \mathrm{id}_P &&: \quad P \to P \end{aligned}$$

where h is a unique morphism in the following diagram.

$$\begin{tikzcd}
 & Q \arrow[r, "\pi_{(X,P,Q)}"] \arrow[d, dashed, "h"'] & P \arrow[d, equal] \\
\mathbb{P} \arrow[d, "q"] & \pi^*_{1qP}P \arrow[r, "\overline{\pi_{1qP}}(P)"] & P \\
\mathbb{B} & \{X\} \arrow[r, "\pi_X"] \arrow[d, "\{\eta^{p\dashv 1}\}"'] & pX \arrow[d, equal, "p\eta=\mathrm{id}_{pX}"] \\
 & \{1pX\} \arrow[r, "\pi_{1pX}"] & pX
\end{tikzcd}$$

Note that η is natural because $\eta^{p\dashv 1}$ is natural and $q : \mathbb{P} \to \mathbb{B}$ is faithful.

The remaining part of the proof is the proof of the triangle identities.

$$\begin{tikzcd}
1 \arrow[r, "\eta_1"] \arrow[dr, equal] & 1\{p \mid q\}1 \arrow[d, "1\epsilon"] \\
 & 1
\end{tikzcd}
\qquad
\begin{tikzcd}
\{p \mid q\} \arrow[r, "\{p|q\}\eta"] \arrow[dr, equal] & \{p \mid q\}1\{p \mid q\} \arrow[d, "\epsilon_{\{p|q\}}"] \\
 & \{p \mid q\}
\end{tikzcd}$$

The right one is easy to verify. The left one follows from the corresponding triangle identity $\mathrm{id} = 1\eta^{p\dashv 1} \circ \eta_1^{p\dashv 1}$ for $p \dashv 1$ where $\epsilon^{p\dashv 1}$ is the identity natural transformation. □

The functor $\{p \mid q\}$ has a comprehension functor.

Lemma 2.2.6. The functor $\{(-)\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ defined by $(X, P, Q) \mapsto Q$ and $(f, g, h) \mapsto h$ is a comprehension functor, i.e., we have an adjunction $1 \dashv \{(-)\}$.

Proof. We define the unit $\eta : \mathrm{Id}_{\mathbb{P}} \to \{1(-)\}$ by

$$\begin{tikzcd}
 & P \arrow[d, dashed, "\eta_P"'] \arrow[dr, equal] & \\
\mathbb{P} \arrow[d, "q"] & \pi^*_{1qP}P \arrow[r, "\overline{\pi_{1qP}}(P)"'] & P \\
\mathbb{B} & qP \arrow[d, "\eta^{1\dashv\{(-)\}}_{qP}"'] \arrow[dr, equal] & \\
 & \{1qP\} \arrow[r, "\pi_{1qP}"'] & qP
\end{tikzcd}$$

and the counit $\epsilon : 1\{(-)\} \to \mathrm{Id}_{\{\mathbb{E}|\mathbb{P}\}}$ by

$$\epsilon_{(X,P,Q)} = (\epsilon_X^{1\dashv\{(-)\}}, \pi_{(X,P,Q)}, \overline{\pi_{1qQ}}(Q)) : (1qQ, Q, \pi^*_{1qQ}Q) \to (X, P, Q).$$

Naturality of η and ϵ can be easily proved.

Lastly, we prove the triangle identities.

$$\begin{tikzcd}
\{(-)\} \arrow[r, "\eta_{\{(-)\}}"] \arrow[dr, equal] & \{1\{(-)\}\} \arrow[d, "\{\epsilon\}"] \\
 & \{(-)\}
\end{tikzcd}
\qquad
\begin{tikzcd}
1(-) \arrow[r, "1\eta"] \arrow[dr, equal] & 1\{1(-)\} \arrow[d, "\epsilon_1"] \\
 & 1(-)
\end{tikzcd}$$

The left one follows by definition. The right one follows from the corresponding triangle identity for $1 \dashv \{(-)\} : \mathbb{E} \to \mathbb{B}$. □

The functor $\{p \mid q\}$ is a full comprehension category with unit.

Lemma 2.2.7. The fibration $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ is a full comprehension category with unit. For each $(X, P, Q) \in \{\mathbb{E} \mid \mathbb{P}\}$, the projection $\pi_{(X,P,Q)} : Q \to P$ is the composition of $\overline{\pi_X}(P) : \pi_X^* P \to P$ and $Q \leq \pi_X^* P$, which is a unique morphism that satisfies $q\pi_{(X,P,Q)} = \pi_X$.

Proof. By Lemma 2.2.5 and Lemma 2.2.6, we have the fibred terminal object $1 : \mathbb{P} \to \{\mathbb{E} \mid \mathbb{P}\}$ and the comprehension functor $\{(-)\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$. The remaining part of the proof is to prove that $\mathcal{P} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}^{\to}$ defined by $(X, P, Q) \mapsto \pi_{(X,P,Q)}$ is full and faithful. Let $(h, g) \in \mathbb{P}^{\to}(\pi_{(X,P,Q)}, \pi_{(X',P',Q')})$.

$$\begin{tikzcd}
Q \arrow[r, "h"] \arrow[d, "\pi_{(X,P,Q)}"'] & Q' \arrow[d, "\pi_{(X',P',Q')}"] \\
P \arrow[r, "g"] & P'
\end{tikzcd}$$

Since $p : \mathbb{E} \to \mathbb{B}$ is a full comprehension category, there exists $f : X \to X'$ that is mapped to $(qh, qg) \in \mathbb{B}^{\to}(\pi_X, \pi_{X'})$ by the functor $\mathbb{E} \to \mathbb{B}^{\to}$.

$$\begin{tikzcd}
\{X\} \arrow[r, "\{f\}"] \arrow[d, "\pi_X"'] & \{X'\} \arrow[d, "\pi_{X'}"] \\
pX \arrow[r, "pf"] & pX'
\end{tikzcd}
\quad = \quad
\begin{tikzcd}
qQ \arrow[r, "qh"] \arrow[d, "q\pi_{(X,P,Q)}"'] & qQ' \arrow[d, "q\pi_{(X',P',Q')}"] \\
qP \arrow[r, "qg"] & qP'
\end{tikzcd}$$

Therefore, we have $(f, g, h) \in \{\mathbb{E} \mid \mathbb{P}\}((X, P, Q), (X', P', Q'))$ and $\mathcal{P}(f, g, h) = (h, g)$. So, $\mathcal{P}$ is full.

Faithfulness of $\mathcal{P}$ follows from faithfulness of the functor $\mathbb{E} \to \mathbb{B}^{\to}$. □

The functor $\{p \mid q\}$ admits strong coproducts.

Lemma 2.2.8. $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ admits strong coproducts.

Proof. For each $(Y, Q, R) \in \{\mathbb{E} \mid \mathbb{P}\}_Q$ and $(f, \mathrm{id}_Q, h) : (Y, Q, R) \to (Y', Q, R')$, $\coprod_{(X,P,Q)}$ is defined by

$$\coprod_{(X,P,Q)} (Y, Q, R) := (\coprod_X Y, P, (\kappa^{-1})^* R) \qquad \coprod_{(X,P,Q)} (f, \mathrm{id}_Q, h) := (\coprod_X f, \mathrm{id}_P, h')$$

where $\kappa = \{\overline{\pi_X}(\coprod_X Y)\} \circ \{\eta^{\coprod_X \dashv \pi_X^*}\} : \{Y\} \to \{\coprod_X Y\}$ and h' is the unique morphism in the following diagram.

$$\begin{tikzcd}
 & (\kappa^{-1})^* R \arrow[r, "\overline{\kappa^{-1}}(R)"] \arrow[d, dashed, "h'"] & R \arrow[d, "h"] \\
\mathbb{P} \arrow[dd, "q"] & (\kappa^{-1})^* R' \arrow[r, "\overline{\kappa^{-1}}(R')"] & R' \\
 & & \\
\mathbb{B} & \{\coprod_X Y\} \arrow[r, "\kappa^{-1}"] \arrow[d, "\{\coprod_X f\}"] & \{Y\} \arrow[d, "\{f\}"] \\
 & \{\coprod_X Y'\} \arrow[r, "\kappa^{-1}"] & \{Y'\}
\end{tikzcd}$$

Note that we have $(\kappa^{-1})^* R \le \pi^*_{\coprod_X Y} P$ because

$$R \le \pi^*_Y Q \le \pi^*_Y \pi^*_X P = \kappa^* \pi^*_{\coprod_X Y} P,$$

and thus, $\coprod_{(X,P,Q)}(Y, Q, R)$ is well-defined.

Adjunction $\coprod_{(X,P,Q)} \dashv \pi^*_{(X,P,Q)}$. Next, we prove that we have an adjunction $\coprod_{(X,P,Q)} \dashv \pi^*_{(X,P,Q)}$. The counit $\epsilon : \coprod_{(X,P,Q)} \pi^*_{(X,P,Q)} \to \mathrm{Id}_{\{\mathbb{E}|\mathbb{P}\}_P}$ is defined by

$$\epsilon_{(Y,P,R)} \coloneqq (\epsilon_Y^{\coprod_X \dashv \pi^*_X}, \mathrm{id}_P, h'')$$

where the morphism h'' over $\{\epsilon_Y\}$ exists because

$$\coprod_{(X,P,Q)} \pi^*_{(X,P,Q)}(Y, P, R) = (\coprod_X \pi^*_X Y, P, (\kappa^{-1})^*(\pi^*_{\pi^*_X Y} Q \wedge \{\overline{\pi_X}(Y)\}^* R))$$

and

$$\kappa^* \{\epsilon_Y\}^* R = \{\overline{\pi_X}(Y)\}^* R \ge \pi^*_{\pi^*_X Y} Q \wedge \{\overline{\pi_X}(Y)\}^* R.$$

The unit $\eta : \mathrm{Id}_{\{\mathbb{E}|\mathbb{P}\}_Q} \to \pi^*_{(X,P,Q)} \coprod_{(X,P,Q)}$ is defined by

$$\eta_{(Y,Q,R)} \coloneqq (\eta_Y^{\coprod_X \dashv \pi^*_X}, \mathrm{id}_Q, \overline{\{\eta_Y^{\coprod_X \dashv \pi^*_X}\}}(\pi^*_{\pi^*_X \coprod_X Y} Q \wedge \{\overline{\pi_X}(\coprod_X Y)\}^* (\kappa^{-1})^* R)).$$

Here, the codomain of $\eta_{(Y,Q,R)} : (Y, Q, R) \to \pi^*_{(X,P,Q)} \coprod_{(X,P,Q)}(Y, Q, R)$ is

$$\pi^*_{(X,P,Q)} \coprod_{(X,P,Q)} (Y, Q, R) = (\pi^*_X \coprod_X Y, Q, \pi^*_{\pi^*_X \coprod_X Y} Q \wedge \{\overline{\pi_X}(\coprod_X Y)\}^* (\kappa^{-1})^* R)$$

and the third component of $\eta_{(Y,Q,R)}$ is the Cartesian lifting of $\{\eta_Y^{\coprod_X \dashv \pi^*_X}\}$ because we have

$$\{\eta_Y^{\coprod_X \dashv \pi^*_X}\}^* \left(\pi^*_{\pi^*_X \coprod_X Y} Q \wedge \{\overline{\pi_X}(\coprod_X Y)\}^* (\kappa^{-1})^* R \right) = \pi^*_Y Q \wedge R = R.$$

Naturality of η and ϵ, and the triangle identities follow from those for $\eta^{\coprod_X \dashv \pi^*_X}$ and $\epsilon^{\coprod_X \dashv \pi^*_X}$.

BC condition. For the BC condition, consider the following diagram where $(f, g, h) : (X, P, Q) \to (X', P', Q')$ is a Cartesian morphism.

$$\begin{tikzcd}
 & \{\mathbb{E} \mid \mathbb{P}\}_P \arrow[d, equal] \\
\{\mathbb{E} \mid \mathbb{P}\}_Q \arrow[ur, "\coprod_{(X,P,Q)}"] & \{\mathbb{E} \mid \mathbb{P}\}_P \arrow[l, "\pi^*_{(X,P,Q)}"] \\
\{\mathbb{E} \mid \mathbb{P}\}_{Q'} \arrow[u, "h^*"] \arrow[d, equal] & \{\mathbb{E} \mid \mathbb{P}\}_{P'} \arrow[u, "g^*"'] \arrow[l, "\pi^*_{(X',P',Q')}"'] \\
\{\mathbb{E} \mid \mathbb{P}\}_{Q'} \arrow[ur, "\coprod_{(X',P',Q')}"'] &
\end{tikzcd}$$

Let $(Y, Q', R) \in \{\mathbb{E} \mid \mathbb{P}\}_{Q'}$. We have

$$\coprod_{(X,P,Q)} h^*(Y, Q', R) = \left(\coprod_X \{f\}^* Y, P, (\kappa^{-1})^*(\pi^*_{\{f\}^*Y} Q \wedge \{\overline{\{f\}}(Y)\}^* R)\right)$$

$$g^* \coprod_{(X',P',Q')} (Y, Q', R) = \left((pf)^* \coprod_{X'} Y, P, \pi^*_{(pf)^* \coprod_X Y} P \wedge \{\overline{pf}(\coprod_{X'} Y)\}^* (\kappa^{-1})^* R\right).$$

To prove that the canonical natural transformation is isomorphic, it suffices to prove the following inequality by Lemma 2.1.2.

$$\begin{aligned} &(\kappa^{-1})^*(\pi^*_{\{f\}^*Y} Q \wedge \{\overline{\{f\}}(Y)\}^* R) \\ &\geq \{\epsilon^{\coprod_X \dashv \pi^*_X} \circ \coprod_X \{f\}^* \eta^{\coprod_{X'} \dashv \pi^*_{X'}}\}^* \left(\pi^*_{(pf)^* \coprod_X Y} P \wedge \{\overline{pf}(\coprod_{X'} Y)\}^* (\kappa^{-1})^* R\right) \end{aligned}$$

This is equivalent to

$$\begin{aligned} &\pi^*_{\{f\}^*Y} Q \wedge \{\overline{\{f\}}(Y)\}^* R \\ &\geq \kappa^* \{\epsilon^{\coprod_X \dashv \pi^*_X} \circ \coprod_X \{f\}^* \eta^{\coprod_{X'} \dashv \pi^*_{X'}}\}^* \left(\pi^*_{(pf)^* \coprod_X Y} P \wedge \{\overline{pf}(\coprod_{X'} Y)\}^* (\kappa^{-1})^* R\right) \end{aligned}$$

We simplify the right-hand side as follows.

$$\begin{aligned} &\kappa^* \{\epsilon^{\coprod_X \dashv \pi^*_X} \circ \coprod_X \{f\}^* \eta^{\coprod_{X'} \dashv \pi^*_{X'}}\}^* \left(\pi^*_{(pf)^* \coprod_X Y} P \wedge \{\overline{pf}(\coprod_{X'} Y)\}^* (\kappa^{-1})^* R\right) \\ &= \{\{f\}^* \eta^{\coprod_{X'} \dashv \pi^*_{X'}}\}^* \{\overline{\pi_X}((pf)^* \coprod_{X'} Y)\}^* \left(\pi^*_{(pf)^* \coprod_X Y} P \wedge \{\overline{pf}(\coprod_{X'} Y)\}^* (\kappa^{-1})^* R\right) \qquad (2.1) \\ &= \pi^*_{\{f\}^*Y} \pi^*_X P \wedge \{\overline{\{f\}}(Y)\}^* R \qquad (2.2) \end{aligned}$$

Here, the first equality (2.1) follows from

$$\begin{tikzcd}
\{f\}^* Y \arrow[r, "\eta^{\coprod_X \dashv \pi^*_X}"] \arrow[d, "\{f\}^* \eta^{\coprod_{X'} \dashv \pi^*_{X'}}"] & \pi^*_X \coprod_X \{f\}^* Y \arrow[r, "\overline{\pi_X}(\dots)"] \arrow[d, "\pi^*_X \coprod_X \{f\}^* \eta"] & \coprod_X \{f\}^* Y \arrow[d, "\coprod_X \{f\}^* \eta"] \\
\{f\}^* \pi^*_{X'} \coprod_{X'} Y \arrow[r, "\eta^{\coprod_X \dashv \pi^*_X}"] \arrow[d, equal] & \pi^*_X \coprod_X \{f\}^* \pi^*_{X'} \coprod_{X'} Y \arrow[r, "\overline{\pi_X}(\dots)"] \arrow[d, equal] & \coprod_X \{f\}^* \pi^*_{X'} \coprod_{X'} Y \arrow[d, equal] \\
\pi^*_X (pf)^* \coprod_{X'} Y \arrow[r, "\eta^{\coprod_X \dashv \pi^*_X}"] \arrow[dr, equal] & \pi^*_X \coprod_X \pi^*_X (pf)^* \coprod_{X'} Y \arrow[r, "\overline{\pi_X}(\dots)"] \arrow[d, "\pi^*_X \epsilon^{\coprod_X \dashv \pi^*_X}"] & \coprod_X \pi^*_X (pf)^* \coprod_{X'} Y \arrow[d, "\epsilon^{\coprod_X \dashv \pi^*_X}"] \\
 & \pi^*_X (pf)^* \coprod_{X'} Y \arrow[r, "\overline{\pi_X}(\dots)"] & (pf)^* \coprod_{X'} Y
\end{tikzcd}$$

(recall the definition of κ and apply rewriting in (2.1) from the top right to the bottom left of the diagram) and the second equality (2.2) follows from the following diagrams.

$$\begin{tikzcd}
\{f\}^*Y \arrow[r, "\{f\}^*\eta"] \arrow[d, "\pi_{\{f\}^*Y}"] & \{f\}^*\pi^*_{X'}\coprod_{X'} Y \arrow[r, equal] \arrow[d, "\pi_{\{f\}^*\pi^*_{X'}\coprod_{X'} Y}"] & \pi^*_X(pf)^*\coprod_{X'} Y \arrow[r, "\overline{\pi_X}(\dots)"] & (pf)^*\coprod_{X'} Y \arrow[d, "\pi_{(pf)^*\coprod_{X'} Y}"] \\
\{X\} \arrow[r, equal] & \{X\} \arrow[rr, "\pi_X"] & & pX
\end{tikzcd}$$

$$\begin{tikzcd}
\{f\}^*Y \arrow[r, "\{f\}^*\eta"] \arrow[d, "\overline{\{f\}}(Y)"] & \{f\}^*\pi^*_{X'}\coprod_{X'} Y \arrow[r, equal] \arrow[d, "\overline{\{f\}}(\dots)"] & \pi^*_X(pf)^*\coprod_{X'} Y \arrow[r, "\overline{\pi_X}(\dots)"] & (pf)^*\coprod_{X'} Y \arrow[d, "\overline{pf}(\dots)"] \\
Y \arrow[r, "\eta"] & \pi^*_{X'}\coprod_{X'} Y \arrow[rr, "\overline{\pi_{X'}}(\dots)"] & & \coprod_{X'} Y
\end{tikzcd}$$

Since $Q = \pi^*_X P \wedge \{f\}^*Q'$ by Lemma 2.2.2 and $R \leq \pi^*_Y Q'$, we have the following inequality.

$$\begin{aligned}
&\pi^*_{\{f\}^*Y} Q \wedge \{\overline{\{f\}}(Y)\}^* R \\
&= \pi^*_{\{f\}^*Y}(\pi^*_X P \wedge \{f\}^*Q') \wedge \{\overline{\{f\}}(Y)\}^* R \\
&= \pi^*_{\{f\}^*Y}\pi^*_X P \wedge \{\overline{\{f\}}(Y)\}^*(\pi^*_Y Q' \wedge R) \\
&\geq \pi^*_{\{f\}^*Y}\pi^*_X P \wedge \{\overline{\{f\}}(Y)\}^* R
\end{aligned}$$

Thus, we conclude that the canonical natural transformation is isomorphic.

Strong sum. Lastly, we prove $\coprod_{(X,P,Q)}$ is a strong sum. Let κ' be the following composite in $\mathbb{P}$.

$$\begin{tikzcd}
\{(Y,Q,R)\} \arrow[r, "\{\eta\}"] & \{\pi^*_{(X,P,Q)}\coprod_{(X,P,Q)(Y,Q,R)}\} \arrow[r, "\{\overline{\pi_{(X,P,Q)}}(\{\coprod_{(X,P,Q)(Y,Q,R)}\})\}"] & \{\coprod_{(X,P,Q)(Y,Q,R)}\}
\end{tikzcd}$$

The morphism $\kappa' : R \to (\kappa^{-1})^*R$ is over κ. We also have $\overline{\kappa}((\kappa^{-1})^*R) : R \to (\kappa^{-1})^*R$ over κ. Since $q : \mathbb{P} \to \mathbb{B}$ is faithful, we have $\kappa' = \overline{\kappa}((\kappa^{-1})^*R)$, and thus κ' is isomorphic by Lemma 2.1.2. □

The functor $\{p \mid q\}$ admits products. The existence of products in $\{p \mid q\}$ requires an additional condition.

Lemma 2.2.9. 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 $\prod_{(X,P,Q)} : \{\mathbb{E} \mid \mathbb{P}\}_Q \to \{\mathbb{E} \mid \mathbb{P}\}_P$ as follows. For each $(Y,Q,R) \in \{\mathbb{E} \mid \mathbb{P}\}_Q$,

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

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

and for each $(f, \mathrm{id}_Q, h) : (Y,Q,R) \to (Y',Q,R')$,

$$\prod_{(X,P,Q)}(f, \mathrm{id}_Q, h) \coloneqq (\prod_X f, \mathrm{id}_P, h_1 \wedge h_2)$$

where h_1 is defined as the unique morphism in

$$\begin{tikzcd}
 & \pi^*_{\prod_X Y} P \arrow[r, "\overline{\pi_{\prod_X Y}}(P)"] \arrow[d, dashed, "h_1"] & P \arrow[d, equal] \\
\mathbb{P} \arrow[dd, "q"] & \pi^*_{\prod_X Y'} P \arrow[r, "\overline{\pi_{\prod_X Y'}}(P)"] & P \\
 & & \\
\mathbb{B} & \{\prod_X Y\} \arrow[r, "\pi_{\prod_X Y}"] \arrow[d, "\{\prod_X f\}"] & pX \arrow[d, equal] \\
 & \{\prod_X Y'\} \arrow[r, "\pi_{\prod_X Y'}"] & pX
\end{tikzcd}$$

and h_2 is defined by the following composite:

$$\begin{array}{c}
\prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X}(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y\}^* R) \\
\rotatebox{-90}{$\leq$} \\
\prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X}(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y\}^* \{f\}^* R') \\
\| \\
\{\prod_X f\}^* \prod_{\pi^*_{\prod_X Y'} X} \sigma^*_{\prod_X Y', X}(\pi^*_{\pi^*_X \prod_X Y'} Q \Rightarrow \{\epsilon_{Y'}\}^* R') \\
\downarrow \overline{\{\prod_X f\}}(\ldots) \\
\prod_{\pi^*_{\prod_X Y'} X} \sigma^*_{\prod_X Y', X}(\pi^*_{\pi^*_X \prod_X Y'} Q \Rightarrow \{\epsilon_{Y'}\}^* R')
\end{array}$$

where the equality in the definition of h_2 follows from the following equations.

$$\begin{aligned}
&\{\prod_X f\}^* \prod_{\pi^*_{\prod_X Y'} X} \sigma^*_{\prod_X Y', X}(\pi^*_{\pi^*_X \prod_X Y'} Q \Rightarrow \{\epsilon_{Y'}^{\pi^*_X \dashv \prod_X}\}^* R') \\
&= \prod_{\pi^*_{\prod_X Y} X} \{\overline{\{\prod_X f\}}(\pi^*_{\prod_X Y'} X)\}^* \sigma^*_{\prod_X Y', X}(\pi^*_{\pi^*_X \prod_X Y'} Q \Rightarrow \{\epsilon_{Y'}^{\pi^*_X \dashv \prod_X}\}^* R') \quad (2.3) \\
&= \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X} \{\pi^*_X \prod_X f\}^* (\pi^*_{\pi^*_X \prod_X Y'} Q \Rightarrow \{\epsilon_{Y'}^{\pi^*_X \dashv \prod_X}\}^* R') \\
&= \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X} \left(\{\pi^*_X \prod_X f\}^* \pi^*_{\pi^*_X \prod_X Y'} Q \Rightarrow \{\pi^*_X \prod_X f\}^* \{\epsilon_{Y'}^{\pi^*_X \dashv \prod_X}\}^* R' \right) \\
&= \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X} \left(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_{Y}^{\pi^*_X \dashv \prod_X}\}^* \{f\}^* R' \right) \quad (2.4)
\end{aligned}$$

The equality (2.3) follows from the BC condition (shown below), and (2.4) follows from naturality of π and ϵ.

$$\begin{tikzcd}
\mathbb{P}_{\{\pi^*_{\prod_X Y}X\}} \arrow[rr, bend left, "\prod_{\pi^*_{\prod_X Y}X}"] \arrow[rr, phantom, "\top"] & & \mathbb{P}_{\{\prod_X Y\}} \arrow[ll, bend left, "\pi^*_{\pi^*_{\prod_X Y}X}"] \\
\mathbb{P}_{\{\pi^*_{\prod_X Y'}X\}} \arrow[u, "\{\overline{\{\prod_X f\}}(\pi^*_{\prod_X Y'}X)\}^*"] \arrow[rr, bend left, "\prod_{\pi^*_{\prod_X Y'}X}"] \arrow[rr, phantom, "\top"] & & \mathbb{P}_{\{\prod_X Y'\}} \arrow[u, "\{\prod_X f\}^*"'] \arrow[ll, bend left, "\pi^*_{\pi^*_{\prod_X Y'}X}"]
\end{tikzcd}$$

Adjunction $\pi^*_{(X,P,Q)} \dashv \prod_{(X,P,Q)}$. Next, we show the existence of an adjunction $\pi^*_{(X,P,Q)} \dashv \prod_{(X,P,Q)}$. We define $\epsilon : \pi^*_{(X,P,Q)} \prod_{(X,P,Q)} \to \mathrm{Id}_{\{\mathbb{E}|\mathbb{P}\}_Q}$ and $\eta : \mathrm{Id}_{\{\mathbb{E}|\mathbb{P}\}_P} \to \prod_{(X,P,Q)} \pi^*_{(X,P,Q)}$ by

$$\begin{aligned}
\epsilon_{(Y,Q,R)} &\coloneqq (\epsilon_Y^{\pi^*_Y \dashv \prod_Y}, \mathrm{id}_Q, h^\epsilon) \\
\eta_{(Y,P,S)} &\coloneqq (\eta_Y^{\pi^*_Y \dashv \prod_Y}, \mathrm{id}_P, h^\eta)
\end{aligned}$$

where h^ϵ and h^η are defined as follows.

The domain of ϵ is

$$\begin{aligned}
&\pi^*_{(X,P,Q)} \prod_{(X,P,Q)} (Y,Q,R) \\
&= (\pi^*_X \prod_X Y, Q, \pi^*_{\pi^*_X \prod_X Y} Q \wedge \{\overline{\pi_X}(\prod_X Y)\}^* \Bigg(\pi^*_{\prod_X Y} P \wedge \\
&\qquad \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X}(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^* R) \Bigg))
\end{aligned}$$

where the third component can be simplified as follows.

$$\begin{aligned}
&\pi^*_{\pi^*_X \prod_X Y} Q \wedge \{\overline{\pi_X}(\prod_X Y)\}^* \Bigg(\pi^*_{\prod_X Y} P \wedge \\
&\qquad \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X}(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^* R) \Bigg) \\
&= \begin{array}{l} \pi^*_{\pi^*_X \prod_X Y} Q \wedge \{\overline{\pi_X}(\prod_X Y)\}^* \pi^*_{\prod_X Y} P \wedge \\ \qquad \{\overline{\pi_X}(\prod_X Y)\}^* \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X}(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^* R) \end{array} \\
&= \begin{array}{l} \pi^*_{\pi^*_X \prod_X Y} Q \wedge \pi^*_{\pi^*_X \prod_X Y} \pi^*_X P \wedge \\ \qquad \sigma^*_{X, \prod_X Y} \pi^*_{\pi^*_{\prod_X Y} X} \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X}(\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^* R) \end{array}
\end{aligned}$$

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

So we define h^ϵ by the following composite.

$$\begin{tikzcd}
\pi^*_{\pi^*_X \prod_X Y} Q \wedge \sigma^*_{X, \prod_X Y} \pi^*_{\pi^*_{\prod_X Y} X} \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X} (\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^* R) \arrow[d, "\pi^*_{\pi^*_X \prod_X Y} Q \wedge \epsilon^{\pi^* \dashv \prod}"] \\
\pi^*_{\pi^*_X \prod_X Y} Q \wedge (\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y\}^* R) \arrow[d, "\mathrm{ev}"] \\
\{\epsilon_Y\}^* R \arrow[d, "\overline{\{\epsilon_Y\}}(R)"] \\
R
\end{tikzcd}$$

The codomain of η is

$$\prod_{(X,P,Q)} \pi^*_{(X,P,Q)} (Y, P, S)$$
$$= (\prod_X \pi^*_X Y, P, \pi^*_{\prod_X \pi^*_X Y} P \wedge$$
$$\prod_{\pi^*_{\prod_X \pi^*_X Y} X} \sigma^*_{\prod_X \pi^*_X Y, X} \left(\pi^*_{\pi^*_X \prod_X \pi^*_X Y} Q \Rightarrow \{\epsilon_{\pi^*_X Y}\}^* (\pi^*_{\pi^*_X Y} Q \wedge \{\overline{\pi_X}(Y)\}^* S) \right))$$

so we define h^η by the composite of the Cartesian lifting of $\{\eta_Y^{\pi^*_X \dashv \prod_X}\}$ and the following inequality.

$$\{\eta_Y\}^* \Bigg(\pi^*_{\prod_X \pi^*_X Y} P \wedge$$
$$\prod_{\pi^*_{\prod_X \pi^*_X Y} X} \sigma^*_{\prod_X \pi^*_X Y, X} \left(\pi^*_{\pi^*_X \prod_X \pi^*_X Y} Q \Rightarrow \{\epsilon_{\pi^*_X Y}\}^* (\pi^*_{\pi^*_X Y} Q \wedge \{\overline{\pi_X}(Y)\}^* S) \right) \Bigg)$$

$$= \pi^*_Y P \wedge \{\eta_Y\}^* \prod_{\pi^*_{\prod_X \pi^*_X Y} X} \sigma^*_{\prod_X \pi^*_X Y, X}$$
$$\left(\pi^*_{\pi^*_X \prod_X \pi^*_X Y} Q \Rightarrow \pi^*_{\pi^*_X \prod_X \pi^*_X Y} Q \wedge \{\epsilon_{\pi^*_X Y}\}^* \{\overline{\pi_X}(Y)\}^* S \right)$$
$$\geq \pi^*_Y P \wedge \{\eta_Y\}^* \prod_{\pi^*_{\prod_X \pi^*_X Y} X} \sigma^*_{\prod_X \pi^*_X Y, X} \{\epsilon_{\pi^*_X Y}\}^* \{\overline{\pi_X}(Y)\}^* S$$
$$= \pi^*_Y P \wedge \prod_{\pi^*_Y X} \{\overline{\{\eta_Y\}}(\pi^*_{\prod_X \pi^*_X Y} X)\}^* \sigma^*_{\prod_X \pi^*_X Y, X} \{\epsilon_{\pi^*_X Y}\}^* \{\overline{\pi_X}(Y)\}^* S$$

$$
\begin{aligned}
&= \pi_Y^* P \wedge \prod_{\pi_Y^* X} \sigma_{Y,X}^* \{\pi_X^* \eta_Y\}^* \{\epsilon_{\pi_X^* Y}\}^* \{\overline{\pi_X}(Y)\}^* S \\
&= \pi_Y^* P \wedge \prod_{\pi_Y^* X} \sigma_{Y,X}^* \sigma_{X,Y}^* \pi_{\pi_Y^* X}^* S \\
&\geq \pi_Y^* P \wedge S \\
&= S
\end{aligned}
$$

Naturality of η and ϵ, and the triangle identities follow from those for $\eta^{\pi_X^* \dashv \prod_X}$ and $\epsilon^{\pi_X^* \dashv \prod_X}$.

BC condition. We prove the BC condition. Let $(f, g, h) : (X, P, Q) \to (X', P', Q')$ be an $\{p \mid q\}$-Cartesian morphism. Consider the following natural transformation.

$$\begin{tikzcd}
 & \{\mathbb{E} \mid \mathbb{P}\}_P \arrow[d, equal] \\
\{\mathbb{E} \mid \mathbb{P}\}_Q \arrow[ur, "\prod_{(X,P,Q)}"] & \{\mathbb{E} \mid \mathbb{P}\}_P \arrow[l, "\pi^*_{(X,P,Q)}"] \\
\{\mathbb{E} \mid \mathbb{P}\}_{Q'} \arrow[u, "h^*"] & \{\mathbb{E} \mid \mathbb{P}\}_{P'} \arrow[u, "g^*"] \arrow[l, "\pi^*_{(X',P',Q')}"'] \\
\{\mathbb{E} \mid \mathbb{P}\}_{Q'} \arrow[u, equal] \arrow[ur, "\prod_{(X',P',Q')}"'] &
\end{tikzcd}$$

For each $(Y, Q', R) \in \{\mathbb{E} \mid \mathbb{P}\}_{Q'}$, the domain of the natural transformation is

$$
\begin{aligned}
&g^* \prod_{(X',P',Q')} (Y, Q', R) \\
&= ((pf)^* \prod_{X'} Y, P, \pi^*_{(pf)^* \prod_{X'} Y} P \wedge \{\overline{pf}(\prod_{X'} Y)\}^* \\
&\qquad \left(\pi^*_{\prod_{X'} Y} P' \wedge \prod_{\pi^*_{\prod_{X'} Y} X'} \sigma^*_{\prod_{X'} Y, X'} (\pi^*_{\pi^*_{X'} \prod_{X'} Y} Q' \Rightarrow \{\epsilon_Y^{\pi^*_{X'} \dashv \prod_{X'}}\}^* R) \right))
\end{aligned}
$$

and the codomain is

$$
\begin{aligned}
&\prod_{(X,P,Q)} h^*(Y, Q', R) \\
&= (\prod_X \{f\}^* Y, P, \pi^*_{\prod_X \{f\}^* Y} P \wedge \prod_{\pi^*_{\prod_X \{f\}^* Y} X} \sigma^*_{\prod_X \{f\}^* Y, X} \\
&\qquad \left(\pi^*_{\pi^*_X \prod_X \{f\}^* Y} Q \Rightarrow \{\epsilon^{\pi^*_X \dashv \prod_X}_{\{f\}^* Y}\}^* (\pi^*_{\{f\}^* Y} Q \wedge \{\overline{\{f\}}(Y)\}^* R) \right)).
\end{aligned}
$$

We can simplify the domain by the following equation.

$$\begin{aligned}
&\pi^*_{(pf)^*\prod_{X'}Y}P \wedge \{\overline{pf}(\prod_{X'}Y)\}^*\pi^*_{\prod_{X'}Y}P' \\
&= \pi^*_{(pf)^*\prod_{X'}Y}P \wedge \pi^*_{(pf)^*\prod_{X'}Y}(pf)^*P' \\
&= \pi^*_{(pf)^*\prod_{X'}Y}(P \wedge (pf)^*P') \\
&= \pi^*_{(pf)^*\prod_{X'}Y}P
\end{aligned}$$

By Lemma 2.1.2, it suffices to prove that the third component of the domain is greater than or equal to the reindexing $\{\theta\}^*$ applied to the third component of the codomain where $\theta = \prod_X\{f\}^*\epsilon^{\pi^*_{X'}\dashv\prod_{X'}} \circ \eta^{\pi^*_X\dashv\prod_X} : (pf)^*\prod_{X'}Y \to \prod_X\{f\}^*Y$ is a vertical isomorphism.

First, note the following equation.

$$\{\theta\}^*\pi^*_{\prod_X\{f\}^*Y}P = \pi^*_{(pf)^*\prod_{X'}Y}P$$

By BC condition for p-products in $\mathbb{P}$ and Lemma 2.1.5, we have

$$\{f\}\prod_{\pi^*_X Y}\sigma^*_{X,Y} = \prod_{\pi^*_{X'}(pf)^*Y}\sigma^*_{X',(pf)^*Y}\{f'\}^*$$

for each $f : X' \to X$ where f' is the same as in Lemma 2.1.5.

$$\begin{aligned}
&\{\theta\}^*\prod_{\pi^*_{\prod_X\{f\}^*Y}X}\sigma^*_{\prod_X\{f\}^*Y,X} = \prod_{\pi^*_{(pf)^*\prod_{X'}Y}X}\sigma^*_{(pf)^*\prod_{X'}Y,X}\{\pi^*_X\theta\}^* \\
&\{\overline{pf}(\prod_{X'}Y)\}^*\prod_{\pi^*_{\prod_{X'}Y}X'}\sigma^*_{\prod_{X'}Y,X'} \\
&= \prod_{\pi^*_{(pf)^*\prod_{X'}Y}(pf)^*X'}\sigma^*_{(pf)^*\prod_{X'}Y,(pf)^*X'}\{\overline{\{\overline{pf}(X')\}}(\pi^*_{X'}\prod_{X'}Y)\}^*
\end{aligned}$$

Since f is Cartesian, there exists $\psi : (pf)^*X' \to X$ such that $\overline{pf}(X') = f \circ \psi$.

$$\begin{aligned}
&\prod_{\pi^*_{(pf)^*\prod_{X'}Y}(pf)^*X'}\sigma^*_{(pf)^*\prod_{X'}Y,(pf)^*X'}\{\overline{\{\overline{pf}(X')\}}(\pi^*_{X'}\prod_{X'}Y)\}^* \\
&= \prod_{\pi^*_{(pf)^*\prod_{X'}Y}(pf)^*X'}\sigma^*_{(pf)^*\prod_{X'}Y,(pf)^*X'}\{\overline{\{f\circ\psi\}}(\pi^*_{X'}\prod_{X'}Y)\}^* \\
&= \prod_{\pi^*_{(pf)^*\prod_{X'}Y}(pf)^*X'}\{\pi^*_{(pf)^*\prod_{X'}Y}\psi\}^*\sigma^*_{(pf)^*\prod_{X'}Y,X}\{\overline{\{f\}}(\pi^*_{X'}\prod_{X'}Y)\}^* \\
&= \prod_{\pi^*_{(pf)^*\prod_{X'}Y}X}\sigma^*_{(pf)^*\prod_{X'}Y,X}\{\overline{\{f\}}(\pi^*_{X'}\prod_{X'}Y)\}^*
\end{aligned}$$

So, it suffices to prove

$$\begin{aligned}
&\prod_{\pi^*_{(pf)^*\prod_{X'}Y}X}\sigma^*_{(pf)^*\prod_{X'}Y,X}\{\overline{\{f\}}(\pi^*_{X'}\prod_{X'}Y)\}^*(\pi^*_{\pi^*_{X'}\prod_{X'}Y}Q' \Rightarrow \{\epsilon^{\pi^*_{X'}\dashv\prod_{X'}}_Y\}^*R) \\
&\geq \pi^*_{(pf)^*\prod_{X'}Y}P \wedge \prod_{\pi^*_{(pf)^*\prod_{X'}Y}X}\sigma^*_{(pf)^*\prod_{X'}Y,X}\{\pi^*_X\theta\}^*\Big(\pi^*_{\pi^*_X\prod_X\{f\}^*Y}Q \Rightarrow \\
&\qquad \{\epsilon^{\pi^*_X\dashv\prod_X}_{\{f\}^*Y}\}^*(\pi^*_{\{f\}^*Y}Q \wedge \{\overline{\{f\}}(Y)\}^*R)\Big)
\end{aligned}$$

where $Q = \pi_X^* P \wedge \{f\}^* Q'$. Observe the following.

$$\begin{aligned}
&\{\overline{\{f\}}(\pi_{X'}^* \prod_{X'} Y)\}^* (\pi^*_{\pi^*_{X'} \prod_{X'} Y} Q' \Rightarrow \{\epsilon_Y^{\pi^*_{X'} \dashv \prod_{X'}}\}^* R) \\
&= \pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} \{f\}^* Q' \Rightarrow \{\{f\}^* \epsilon^{\pi^*_{X'} \dashv \prod_{X'}}\}^* \{\overline{\{f\}}(Y)\}^* R
\end{aligned}$$

$$\begin{aligned}
&\{\pi_X^* \theta\}^* \left(\pi^*_{\pi^*_X \prod_X \{f\}^* Y} Q \Rightarrow \{\epsilon^{\pi^*_X \dashv \prod_X}_{\{f\}^* Y}\}^* (\pi^*_{\{f\}^* Y} Q \wedge \{\overline{\{f\}}(Y)\}^* R)\right) \\
&= \pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} Q \Rightarrow (\pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} Q \wedge \{\{f\}^* \epsilon^{\pi^*_{X'} \dashv \prod_{X'}}\}^* \{\overline{\{f\}}(Y)\}^* R)
\end{aligned}$$

$$\begin{aligned}
&\pi^*_{(pf)^* \prod_{X'} Y} P \wedge \prod_{\pi^*_{(pf)^* \prod_{X'} Y} X} \sigma^*_{(pf)^* \prod_{X'} Y, X} (\dots) \\
&\leq \prod_{\pi^*_{(pf)^* \prod_{X'} Y} X} \sigma^*_{(pf)^* \prod_{X'} Y, X} \sigma^*_{X, (pf)^* \prod_{X'} Y} \pi^*_{\pi^*_{(pf)^* \prod_{X'} Y} X} \Big(\pi^*_{(pf)^* \prod_{X'} Y} P \wedge \\
&\qquad\qquad \prod_{\pi^*_{(pf)^* \prod_{X'} Y} X} \sigma^*_{(pf)^* \prod_{X'} Y, X} (\dots)\Big) \\
&\leq \prod_{\pi^*_{(pf)^* \prod_{X'} Y} X} \sigma^*_{(pf)^* \prod_{X'} Y, X} \left(\pi^*_{\pi^*_X (pf)^* \prod_{X'} Y} \pi^*_X P \wedge (\dots)\right)
\end{aligned}$$

Now it suffices to prove

$$\begin{aligned}
&\pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} \{f\}^* Q' \Rightarrow \{\{f\}^* \epsilon^{\pi^*_{X'} \dashv \prod_{X'}}\}^* \{\overline{\{f\}}(Y)\}^* R \\
&\geq \pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} \pi^*_X P \wedge \Big(\pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} Q \Rightarrow \\
&\qquad\qquad (\pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} Q \wedge \{\{f\}^* \epsilon^{\pi^*_{X'} \dashv \prod_{X'}}\}^* \{\overline{\{f\}}(Y)\}^* R)\Big)
\end{aligned}$$

which is equivalent to

$$\begin{aligned}
&\{\{f\}^* \epsilon^{\pi^*_{X'} \dashv \prod_{X'}}\}^* \{\overline{\{f\}}(Y)\}^* R \\
&\geq \pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} \{f\}^* Q' \wedge \pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} \pi^*_X P \wedge \\
&\quad \left(\pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} Q \Rightarrow (\pi^*_{\{f\}^* \pi^*_{X'} \prod_{X'} Y} Q \wedge \{\{f\}^* \epsilon^{\pi^*_{X'} \dashv \prod_{X'}}\}^* \{\overline{\{f\}}(Y)\}^* R)\right).
\end{aligned}$$

This inequality follows from $Q = \pi_X^* P \wedge \{f\}^* Q'$ and the evaluation morphism of exponentials. □

The functor $\{p \mid q\}$ is a SSCompC. As a result of above lemmas, we get a lifting of SCCompCs over $p : \mathbb{E} \to \mathbb{B}$.

Theorem 2.2.10. 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 2.2.7,2.2.8,2.2.9. 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 $(\mathcal{P}^*(q^{\rightarrow}), q)$ preserves the structure of CCompCs. □

Example 2.2.11. Consider the simple fibration $\mathsf{s}_{\mathbf{Set}} : \mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$ and the subobject fibration $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$. 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 and coproducts are given by

$$\begin{aligned}\prod_{((I,X),P,Q)} ((I \times X, Y), Q, R) = \big((I, X \Rightarrow Y), P, \{(i, f) \in I \times (X \Rightarrow Y) \mid \\ i \in P \wedge \forall x \in X, (i, x) \in Q \implies ((i, x), f(x)) \in R\}\big) \qquad (2.5)\\ \coprod_{((I,X),P,Q)} ((I \times X, Y), Q, R) = \big((I, X \times Y), P, \{(i, (x, y)) \mid ((i, x), y) \in R\}\big).\end{aligned}$$

Example 2.2.12. 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 2.2.10 to erel and the simple fibration $\mathsf{s}_{\mathbf{Set}} : \mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$, then products are defined similarly to Example 2.2.11.

Example 2.2.13. We can construct a SCCompC for relational verification as follows. Let $p : \mathbb{E} \to \mathbb{B}$ be a SCCompC and $q : \mathbb{P} \to \mathbb{B}$ be a fibred ccc. We have a SCCompC $p^2 : \mathbb{E}^2 \to \mathbb{B}^2$. Here, each of the CCompC structures of $p^2 : \mathbb{E}^2 \to \mathbb{B}^2$ are defined component-wise from the corresponding structure of $p : \mathbb{E} \to \mathbb{B}$. We also consider the fibration $\mathbf{BRel}(q) : \mathbf{BRel}(\mathbb{P}) \to \mathbb{B}^2$ defined by the change-of-base along $\times : \mathbb{B}^2 \to \mathbb{B}$.

$$\begin{tikzcd}
\mathbf{BRel}(\mathbb{P}) \arrow[r] \arrow[d, "\mathbf{BRel}(q)"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbb{P} \arrow[d, "q"] \\
\mathbb{B}^2 \arrow[r, "\times"] & \mathbb{B}
\end{tikzcd}$$

The fibration $\mathbf{BRel}(q) : \mathbf{BRel}(\mathbb{P}) \to \mathbb{B}^2$ is posetal and a fibred ccc because the change-of-base functor $(\times)^* : \mathbf{Fib}_{\mathbb{B}} \to \mathbf{Fib}_{\mathbb{B}^2}$ is a 2-functor that preserves finite products. Therefore, we can apply Theorem 2.2.10 to the combination of $p^2 : \mathbb{E}^2 \to \mathbb{B}^2$ and $\mathbf{BRel}(q) : \mathbf{BRel}(\mathbb{P}) \to \mathbb{B}^2$ if $\mathbf{BRel}(q)$ has p^2-products.

For example, when p is the simple fibration $\mathsf{s}_{\mathbf{Set}} : \mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$ and q is the subobject fibration $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$, then we obtain a SCCompC $\{\mathsf{s}^2_{\mathbf{Set}} \mid \mathbf{BRel}(\mathsf{sub}_{\mathbf{Set}})\} : \{\mathbf{s}(\mathbf{Set})^2 \mid \mathbf{BRel}(\mathbf{Sub}(\mathbf{Set}))\} \to \mathbf{BRel}(\mathbf{Sub}(\mathbf{Set}))$. An object in the total category is $((I_1, X_1), (I_2, X_2), P, Q)$ where $(I_1, X_1), (I_2, X_2) \in \mathbf{s}(\mathbf{Set})$, $P \subseteq I_1 \times I_2$, and $Q \subseteq I_1 \times X_1 \times I_2 \times X_2$.

Example 2.2.14. Consider the family fibration $\mathsf{fam}_{\mathbf{Set}} : \mathbf{Fam}(\mathbf{Set}) \to \mathbf{Set}$ [53, 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 (2.5) for dependent functions.

2.2.2 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 [49, Prop. 4.5.8].

Lemma 2.2.15. If

- $p : \mathbb{E} \to \mathbb{B}$ has strong fibred coproducts
- for each $X, Y \in \mathbb{E}_I$, $X', Y' \in \mathbb{E}_{I'}$, $u : I \to I'$ and Cartesian lifting $f : X \to X'$ and $g : Y \to Y'$ over u, the following two squares are pullbacks

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

  where $\iota_1 : X \to X + Y$ and $\iota_2 : Y \to X + Y$ are coprojections
- $q : \mathbb{P} \to \mathbb{B}$ is a fibred distributive category
- for each $X, Y \in \mathbb{E}_I$ and $Z \in \mathbb{E}_{\{X+Y\}}$, q has cocartesian lifting of
  - $\{\iota_1\} : \{X\} \to \{X + Y\}$
  - $\{\iota_2\} : \{Y\} \to \{X + Y\}$
  - $\{\overline{\{\iota_1\}}(Z)\} : \{\{\iota_1\}^* Z\} \to \{Z\}$
  - $\{\overline{\{\iota_2\}}(Z)\} : \{\{\iota_2\}^* Z\} \to \{Z\}$

  that satisfy the Beck-Chevalley condition for each pullback squares and the Frobenius condition

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 coproducts.

Proof. We define fibred coproducts by

$$(X, P, Q) + (Y, P, R) \coloneqq (X + Y, P, \{\iota_1\}_! Q \vee \{\iota_2\}_! R)$$

and coprojections by

$$\begin{aligned}
&(\iota_1, \mathrm{id}_P, \iota_1 \circ \underline{\{\iota_1\}}(Q)) : (X, P, Q) \to (X, P, Q) + (Y, P, R) \\
&(\iota_2, \mathrm{id}_P, \iota_2 \circ \underline{\{\iota_2\}}(Q)) : (Y, P, R) \to (X, P, Q) + (Y, P, R).
\end{aligned}$$

For each $(f_1, \mathrm{id}_P, h_1) : (X, P, Q) \to (Z, P, S)$ and $(f_2, \mathrm{id}_P, h_2) : (Y, P, R) \to (Z, P, S)$, we define the cotupling by

$$\begin{aligned}
[(f_1, \mathrm{id}_P, h_1), (f_2, \mathrm{id}_P, h_2)] &= ([f_1, f_2], \mathrm{id}_P, [h_1', h_2']) \\
&\qquad : (X, P, Q) + (Y, P, R) \to (Z, P, S)
\end{aligned}$$

where h_1' and h_2' are defined as follows.

$$\begin{tikzcd}
 & & S & & S \\
\mathbb{P} \arrow[d, "q"] & Q \arrow[ur, "h_1"] \arrow[r, "\underline{\{\iota_1\}}(Q)"'] & \{\iota_1\}_!Q \arrow[u, dashed, "h'_1"'] & R \arrow[ur, "h_2"] \arrow[r, "\underline{\{\iota_2\}}(R)"'] & \{\iota_2\}_!R \arrow[u, dashed, "h'_2"'] \\
\mathbb{B} & & \{Z\} & & \{Z\} \\
 & \{X\} \arrow[ur, "\{f_1\}"] \arrow[r, "\{\iota_1\}"'] & \{X+Y\} \arrow[u, "\{[f_1,f_2]\}"'] & \{Y\} \arrow[ur, "\{f_2\}"] \arrow[r, "\{\iota_2\}"'] & \{X+Y\} \arrow[u, "\{[f_1,f_2]\}"']
\end{tikzcd}$$

It is easy to verify that this definition satisfies universal property of coproducts in $\{\mathbb{E} \mid \mathbb{P}\}_P$.

Fibred coproduct. Next, we prove that coproducts are preserved by reindexing functors. Let $f : P \to P'$ be a morphism in $\mathbb{P}$ and $(X, P', Q), (Y, P', R) \in \{\mathbb{E} \mid \mathbb{P}\}_{P'}$. We have a canonical morphism

$$[f^*\iota_1, f^*\iota_2] : f^*(X, P', Q) + f^*(Y, P', Q) \to f^*((X, P', Q) + (Y, P', R))$$

where

$$\begin{aligned}
&f^*(X, P', Q) + f^*(Y, P', Q) \\
&= ((qf)^*X + (qf)^*Y, \quad P', \\
&\qquad\qquad \{\iota_1\}_!(\pi^*_{(qf)^*X}P' \wedge \{\overline{qf}(X)\}^*Q) \vee \{\iota_2\}_!(\pi^*_{(qf)^*X}P' \wedge \{\overline{qf}(X)\}^*R)) \\
&f^*((X, P', Q) + (Y, P', R)) \\
&= ((qf)^*(X+Y), P', \pi^*_{(qf)^*(X+Y)}P' \wedge \{\overline{qf}(X+Y)\}^*(\{\iota_1\}_!Q \vee \{\iota_2\}_!R)).
\end{aligned}$$

The canonical morphism is isomorphic by Lemma 2.1.2 and the following equation.

$$\begin{aligned}
&\{[(qf)^*\iota_1, (qf)^*\iota_2]\}^*\big(\pi^*_{(qf)^*(X+Y)}P' \wedge \{\overline{qf}(X+Y)\}^*(\{\iota_1\}_!Q \vee \{\iota_2\}_!R)\big) \\
&= \pi^*_{(qf)^*X+(qf)^*Y}P' \wedge \{\overline{qf}(X) + \overline{qf}(Y)\}^*(\{\iota_1\}_!Q \vee \{\iota_2\}_!R) \\
&= \pi^*_{(qf)^*X+(qf)^*Y}P' \wedge \big(\{\iota_1\}_!\{\overline{qf}(X)\}^*Q \vee \{\iota_2\}_!\{\overline{qf}(Y)\}^*R\big) \\
&= \big(\pi^*_{(qf)^*X+(qf)^*Y}P' \wedge \{\iota_1\}_!\{\overline{qf}(X)\}^*Q\big)\vee \\
&\qquad\qquad \big(\pi^*_{(qf)^*X+(qf)^*Y}P' \wedge \{\iota_2\}_!\{\overline{qf}(Y)\}^*R\big) \\
&= \{\iota_1\}_!\big(\{\iota_1\}^*\pi^*_{(qf)^*X+(qf)^*Y}P' \wedge \{\overline{qf}(X)\}^*Q\big)\vee \\
&\qquad\qquad \{\iota_2\}_!\big(\{\iota_2\}^*\pi^*_{(qf)^*X+(qf)^*Y}P' \wedge \{\overline{qf}(Y)\}^*R\big) \\
&= \{\iota_1\}_!\big(\pi^*_{(qf)^*X}P' \wedge \{\overline{qf}(X)\}^*Q\big) \vee \{\iota_2\}_!\big(\pi^*_{(qf)^*Y}P' \wedge \{\overline{qf}(Y)\}^*R\big)
\end{aligned}$$

Strong fibred coproduct. Finally, we prove that fibred coproducts of $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ are strong. That is, the functor

$$\{\mathbb{E} \mid \mathbb{P}\}_{\{\iota_1\}_!Q \vee \{\iota_2\}_!R} \xrightarrow{\langle \{\iota_1\}^*, \{\iota_2\}^* \rangle} \{\mathbb{E} \mid \mathbb{P}\}_Q \times \{\mathbb{E} \mid \mathbb{P}\}_R$$

is full and faithful.

The functor $\langle \{\iota_1\}^*, \{\iota_2\}^* \rangle$ is faithful: if $\langle \{\iota_1\}^*, \{\iota_2\}^* \rangle$ maps two morphism $(f_1, \mathrm{id}, h_1), (f_2, \mathrm{id}, h_2) : (Z, \{\iota_1\}_! Q \vee \{\iota_2\}_! R, S) \to (Z', \{\iota_1\}_! Q \vee \{\iota_2\}_! R, S')$ to the same pair of morphisms, then $f_1 = f_2$ follows because $p : \mathbb{E} \to \mathbb{B}$ has strong fibred coproducts, and $h_1 = h_2$ also follows because $q : \mathbb{P} \to \mathbb{B}$ is faithful.

For fullness, given

- $(X, P, Q), (Y, P, R) \in \{\mathbb{E} \mid \mathbb{P}\}_P$,
- two objects over $\{(X, P, Q) + (Y, P, R)\}$

  $$(Z_1, \{\iota_1\}_! Q \vee \{\iota_2\}_! R, S_1), (Z_2, \{\iota_1\}_! Q \vee \{\iota_2\}_! R, S_2) \in \{\mathbb{E} \mid \mathbb{P}\}_{\{\iota_1\}_! Q \vee \{\iota_2\}_! R},$$

- and a pair of morphisms in $\{\mathbb{E} \mid \mathbb{P}\}_Q$ and $\{\mathbb{E} \mid \mathbb{P}\}_R$

  $$\begin{aligned}
  (f_1, \mathrm{id}_Q, h_1) &: (\{\iota_1\}^* Z_1, Q, \pi^*_{\{\iota_1\}^* Z_1} Q \wedge \{\overline{\{\iota_1\}}(Z_1)\}^* S_1) \\
  &\qquad \to (\{\iota_1\}^* Z_2, Q, \pi^*_{\{\iota_1\}^* Z_2} Q \wedge \{\overline{\{\iota_1\}}(Z_2)\}^* S_2) \\
  (f_2, \mathrm{id}_R, h_2) &: (\{\iota_2\}^* Z_1, R, \pi^*_{\{\iota_2\}^* Z_1} R \wedge \{\overline{\{\iota_2\}}(Z_1)\}^* S_1) \\
  &\qquad \to (\{\iota_2\}^* Z_2, R, \pi^*_{\{\iota_2\}^* Z_2} R \wedge \{\overline{\{\iota_2\}}(Z_2)\}^* S_2),
  \end{aligned}$$

we prove that there exists a morphism

$$(f, \mathrm{id}, h) : (Z_1, \{\iota_1\}_! Q \vee \{\iota_2\}_! R, S_1) \to (Z_2, \{\iota_1\}_! Q \vee \{\iota_2\}_! R, S_2)$$

such that $(f_1, \mathrm{id}, g_1) = \{\iota_1\}^*(f, \mathrm{id}, h)$ and $(f_2, \mathrm{id}, g_2) = \{\iota_2\}^*(f, \mathrm{id}, h)$. Since $\langle \{\iota_1\}^*, \{\iota_2\}^* \rangle : \mathbb{E}_{\{X+Y\}} \to \mathbb{E}_{\{X\}} \times \mathbb{E}_{\{Y\}}$ is full, there exists $f : Z_1 \to Z_2$ such that $\{\iota_1\}^* f = f_1$ and $\{\iota_2\}^* f = f_2$. To prove the existence of the morphism h over $\{f\}$, it suffices to prove $S_1 \leq \{f\}^* S_2$.

$$\begin{aligned}
S_1 &= S_1 \wedge \pi^*_{Z_1}(\{\iota_1\}_! Q \vee \{\iota_2\}_! R) \\
&= (S_1 \wedge \pi^*_{Z_1}\{\iota_1\}_! Q) \vee (S_1 \wedge \pi^*_{Z_1}\{\iota_2\}_! R) \\
&= \big(S_1 \wedge \{\overline{\{\iota_1\}}(Z_1)\}_! \pi^*_{\{\iota_1\}^* Z_1} Q\big) \vee \big(S_1 \wedge \{\overline{\{\iota_2\}}(Z_1)\}_! \pi^*_{\{\iota_2\}^* Z_1} R\big) \\
&= \{\overline{\{\iota_1\}}(Z_1)\}_! \left(\{\overline{\{\iota_1\}}(Z_1)\}^* S_1 \wedge \pi^*_{\{\iota_1\}^* Z_1} Q\right) \vee \\
&\qquad\qquad \{\overline{\{\iota_2\}}(Z_1)\}_! \left(\{\overline{\{\iota_2\}}(Z_1)\}^* S_1 \wedge \pi^*_{\{\iota_2\}^* Z_1} R\right) \\
&\leq \{\overline{\{\iota_1\}}(Z_1)\}_! \{f_1\}^* \left(\{\overline{\{\iota_1\}}(Z_2)\}^* S_2 \wedge \pi^*_{\{\iota_1\}^* Z_2} Q\right) \vee \\
&\qquad\qquad \{\overline{\{\iota_2\}}(Z_1)\}_! \{f_2\}^* \left(\{\overline{\{\iota_2\}}(Z_2)\}^* S_2 \wedge \pi^*_{\{\iota_2\}^* Z_2} R\right) \\
&\leq \{\overline{\{\iota_1\}}(Z_1)\}_! \{f_1\}^* \{\overline{\{\iota_1\}}(Z_2)\}^* S_2 \vee \{\overline{\{\iota_2\}}(Z_1)\}_! \{f_2\}^* \{\overline{\{\iota_2\}}(Z_2)\}^* S_2 \\
&= \{\overline{\{\iota_1\}}(Z_1)\}_! \{\overline{\{\iota_1\}}(Z_1)\}^* \{f\}^* S_2 \vee \{\overline{\{\iota_2\}}(Z_1)\}_! \{\overline{\{\iota_2\}}(Z_1)\}^* \{f\}^* S_2 \\
&\leq \{f\}^* S_2
\end{aligned}$$

□

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

Example 2.2.16. 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 2.2.11). This combination satisfies four conditions in Lemma 2.2.15. Fibred coproducts in $\{\mathbf{s}(\mathbf{Set}) \mid \mathbf{Sub}(\mathbf{Set})\} \to \mathbf{Sub}(\mathbf{Set})$ are defined as follows.

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

2.3 Lifting Monads on SCCompCs

In this section, we consider computational effects in dependent refinement type systems.

Definition 2.3.1 (monad lifting)**.** Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration and (T, η, μ) be a monad on $\mathbb{B}$. A *lifting* (or *monad lifting*) of T is a monad $(\dot{T}, \dot{\eta}, \dot{\mu})$ on $\mathbb{E}$ such that $p\dot{T} = Tp$, $p\dot{\eta} = \eta_p$, and $p\dot{\mu} = \mu_p$. A *fibred lifting* of T is a lifting $(\dot{T}, \dot{\eta}, \dot{\mu})$ of T such that $\dot{T}$ is a fibred functor, and a *Cartesian lifting* is a fibred lifting $(\dot{T}, \dot{\eta}, \dot{\mu})$ of T such that each component of $\dot{\eta}$ and $\dot{\mu}$ is a Cartesian morphism.

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 2.2.10. We are going to 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 2.3.2) $\theta : \{\hat{T}(-)\} \to T\{-\}$. 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 [8]. 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 2.3.2 (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 2.3.3. 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 [8] of T along $q : \mathbb{P} \to \mathbb{B}$. If

$$\pi^*_{\hat{T}X} P \wedge \theta^*_X \dot{T} Q \le \theta^*_X \dot{T}(\pi^*_X P \wedge Q) \tag{2.6}$$

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 2.2.10 is a fibred monad morphism from S to $\hat{T}$.

$$\begin{tikzcd}
\mathbb{E} \arrow[loop right, "\hat{T}"] \arrow[d, "p"] & \mathbb{P} \arrow[loop right, "\dot{T}"] \arrow[d, "q"] \\
\mathbb{B} & \mathbb{B} \arrow[loop right, "T"]
\end{tikzcd}$$

Proof. The functor $S : \{\mathbb{E} \mid \mathbb{P}\} \to \{\mathbb{E} \mid \mathbb{P}\}$ is defined by

$$S(X, P, Q) \coloneqq (\hat{T}X, P, \pi^*_{\hat{T}X} P \wedge \theta^*_X \dot{T}Q) \qquad S(f, g, h) \coloneqq (\hat{T}f, g, h_0)$$

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

for each $(X, P, Q), (X', P', Q') \in \{\mathbb{E} \mid \mathbb{P}\}$ and $(f, g, h) : (X, P, Q) \to (X', P', Q')$ where $h_0 : \pi^*_{\hat{T}X} P \wedge \theta^*_X \dot{T}Q \to \pi^*_{\hat{T}X'} P' \wedge \theta^*_{X'} \dot{T}Q'$ is defined by combining the two dashed arrows in the following diagram.

$$\begin{tikzcd}
 & \pi^*_{\hat{T}X} P \arrow[r, "\overline{\pi_{\hat{T}X}}(P)"] \arrow[d, dashed] & P \arrow[d, "g"] & \theta^*_X \dot{T}Q \arrow[r, "\overline{\theta_X}(\dot{T}Q)"] \arrow[d, dashed] & \dot{T}Q \arrow[d, "\dot{T}h"] \\
\mathbb{P} \arrow[dd, "q"] & \pi^*_{\hat{T}X'} P' \arrow[r, "\overline{\pi_{\hat{T}X'}}(P')"] & P' & \theta^*_{X'} \dot{T}Q' \arrow[r, "\overline{\theta_{X'}}(\dot{T}Q')"] & \dot{T}Q' \\
 & & & & \\
\mathbb{B} & \{\hat{T}X\} \arrow[r, "\pi_{\hat{T}X}"] \arrow[d, "\{\hat{T}f\}"] & pX \arrow[d, "pf"] & \{\hat{T}X\} \arrow[r, "\theta_X"] \arrow[d, "\{\hat{T}f\}"] & T\{X\} \arrow[d, "T\{f\}"] \\
 & \{\hat{T}X'\} \arrow[r, "\pi_{\hat{T}X'}"] & pX' & \{\hat{T}X'\} \arrow[r, "\theta_{X'}"] & T\{X'\}
\end{tikzcd}$$

The unit is defined by

$$\eta_{(X,P,Q)} \coloneqq (\eta^{\hat{T}}_X, \mathrm{id}_P, \langle h_1, h_2 \rangle) : (X, P, Q) \to (\hat{T}X, P, \pi^*_{\hat{T}X} P \wedge \theta^*_X \dot{T}Q)$$

where h_1 and h_2 are unique morphisms in the following diagram.

$$\begin{tikzcd}
Q \arrow[r, "\pi_{(X,P,Q)}"] \arrow[d, dashed, "h_1"] & P \arrow[d, equal] & & Q \arrow[d, dashed, "h_2"] \arrow[dr, "\eta^{\dot{T}}_Q"] & \\
\pi^*_{\hat{T}X} P \arrow[r, "\overline{\pi_{\hat{T}X}}(P)"] & P & \mathbb{P} \arrow[d, "q"] & \theta^*_X \dot{T}Q \arrow[r, "\overline{\theta_X}(\dot{T}Q)"'] & \dot{T}Q \\
\{X\} \arrow[r, "\pi_X"] \arrow[d, "\{\eta^{\hat{T}}_X\}"] & pX \arrow[d, equal, "p\eta^{\hat{T}}_X"] & \mathbb{B} & \{X\} \arrow[d, "\{\eta^S_X\}"'] \arrow[dr, "\eta^T_{\{X\}}"] & \\
\{\hat{T}X\} \arrow[r, "\pi_{\hat{T}X}"] & pX & & \{\hat{T}X\} \arrow[r, "\theta_X"] & T\{X\}
\end{tikzcd}$$

The multiplication is defined by

$$\mu_{(X,P,Q)} \coloneqq (\mu^{\hat{T}}_X, \mathrm{id}_P, h_3 \wedge (h_4 \circ \theta^*_{\hat{T}X} \dot{T}\pi_2)) : S^2(X, P, Q) \to S(X, P, Q)$$

where the domain of $\mu_{(X,P,Q)}$ is given by

$$S^2(X,P,Q) = (\hat{T}^2X, P, \pi^*_{\hat{T}^2X}P \wedge \theta^*_{\hat{T}X}\dot{T}(\pi^*_{\hat{T}X}P \wedge \theta^*_X\dot{T}Q))$$

and thus, h_3 and h_4 are defined by the following diagram.

$$\begin{tikzcd}
\pi^*_{\hat{T}^2X}P \arrow[r, "\overline{\pi_{\hat{T}^2X}}(P)"] \arrow[d, dashed, "h_3"] & P \arrow[d, equal] \\
\pi^*_{\hat{T}X}P \arrow[r, "\overline{\pi_{\hat{T}X}}(P)"] & P
\end{tikzcd}
\qquad
\begin{tikzcd}
\theta^*_{TX}\dot{T}\theta^*_X\dot{T}Q \arrow[r, "\overline{\theta_{TX}}(\dot{T}\theta^*_X\dot{T}Q)"] \arrow[d, dashed, "h_4"] & \dot{T}\theta^*_X\dot{T}Q \arrow[r, "\dot{T}\overline{\theta_X}(\dot{T}Q)"] & \dot{T}^2Q \arrow[d, "\mu^{\dot{T}}_Q"] \\
\theta^*_X\dot{T}Q \arrow[rr, "\overline{\theta_X}(\dot{T}Q)"] & & \dot{T}Q
\end{tikzcd}$$

$$\begin{tikzcd}
\{\hat{T}^2X\} \arrow[r, "\pi_{\hat{T}^2X}"] \arrow[d, "\{\mu^{\hat{T}}_X\}"] & pX \arrow[d, equal, "p\mu^{\hat{T}}_X"] \\
\{\hat{T}X\} \arrow[r, "\pi_{\hat{T}^2X}"] & pX
\end{tikzcd}
\qquad
\begin{tikzcd}
\{\hat{T}^2X\} \arrow[r, "\theta_{\hat{T}X}"] \arrow[d, "\{\mu^{\hat{T}}_X\}"] & T\{\hat{T}X\} \arrow[r, "T\theta_X"] & T^2\{X\} \arrow[d, "\mu^T_{\{X\}}"] \\
\{\hat{T}X\} \arrow[rr, "\theta_X"] & & T\{X\}
\end{tikzcd}$$

The monad laws follow from those for $\hat{T}$.

The monad S is fibred. We prove S is a fibred functor. Let $(f,g,h) : (X,P,Q) \to (X',P',Q')$ be an $\{p \mid q\}$-Cartesian morphism. By Lemma 2.2.2, $f : X \to X'$ is p-Cartesian and $Q = \pi^*_XP \wedge \{f\}^*Q'$. To prove that $S(f,g,h)$ is also $\{p \mid q\}$-Cartesian, it suffices to prove that $\hat{T}f$ is p-Cartesian and $\pi^*_{\hat{T}X}P \wedge \theta^*_X\dot{T}Q = \pi^*_{\hat{T}X}P \wedge \{\hat{T}f\}^*(\pi^*_{\hat{T}X'}P' \wedge \theta^*_{X'}\dot{T}Q')$. The former follows because $\hat{T}$ is a fibred functor. For the latter, the following three inequalities obviously hold.

$$\begin{aligned}
\pi^*_{\hat{T}X}P \wedge \theta^*_X\dot{T}Q &\leq \pi^*_{\hat{T}X}P \\
\pi^*_{\hat{T}X}P \wedge \theta^*_X\dot{T}Q &\leq \{\hat{T}f\}^*(\pi^*_{\hat{T}X'}P' \wedge \theta^*_{X'}\dot{T}Q') \\
\pi^*_{\hat{T}X}P &\geq \pi^*_{\hat{T}X}P \wedge \{\hat{T}f\}^*(\pi^*_{\hat{T}X'}P' \wedge \theta^*_{X'}\dot{T}Q')
\end{aligned}$$

So, it suffices to prove $\theta^*_X\dot{T}Q \geq \pi^*_{\hat{T}X}P \wedge \{\hat{T}f\}^*(\pi^*_{\hat{T}X'}P' \wedge \theta^*_{X'}\dot{T}Q')$. The right-hand side can be simplified as follows.

$$\begin{aligned}
&\pi^*_{\hat{T}X}P \wedge \{\hat{T}f\}^*(\pi^*_{\hat{T}X'}P' \wedge \theta^*_{X'}\dot{T}Q') \\
&= \pi^*_{\hat{T}X}P \wedge \{\hat{T}f\}^*\pi^*_{\hat{T}X'}P' \wedge \{\hat{T}f\}^*\theta^*_{X'}\dot{T}Q' \\
&= \pi^*_{\hat{T}X}P \wedge \pi^*_{\hat{T}X}(pf)^*P' \wedge \theta^*_X(T\{f\})^*\dot{T}Q' \\
&= \pi^*_{\hat{T}X}P \wedge \theta^*_X(T\{f\})^*\dot{T}Q'
\end{aligned}$$

Since we have $Q = \pi^*_XP \wedge \{f\}^*Q'$, what we need to show is

$$\pi^*_{\hat{T}X}P \wedge \theta^*_X(T\{f\})^*\dot{T}Q' \leq \theta^*_X\dot{T}(\pi^*_XP \wedge \{f\}^*Q'),$$

which follows from (2.6) and the fibredness of $\dot{T}$. □

Example 2.3.4. Any strong monad T on a CCC $\mathbb{B}$ gives rise to a split fibred monad $\hat{T}$ on the simple fibration $\mathbf{s}_{\mathbb{B}} : \mathbf{s}(\mathbb{B}) \to \mathbb{B}$ (actually, there is a one-to-one correspondence [53, 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 [84, Proposition 3.4], we can prove that (2.6) 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 2.3.3 from $\dot{T}_1$ and $\dot{T}_2$ are given by

$$\begin{aligned} &((I, X), P, Q) \mapsto \big((I, X + \{*\}), P, \{(i, x) \mid (i \in P \land x = *) \lor (i, x) \in Q\}\big) \\ &((I, X), P, Q) \mapsto \big((I, X + \{*\}), P, \{(i, x) \mid (i, x) \in Q\}\big) \end{aligned}$$

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

Example 2.3.5. 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 (2.6) 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 2.3.3 is given by

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

2.4 Soundness

We consider a concrete dependent refinement type system with computational effects and define sound semantics to show that the SCCompC defined in Theorem 2.2.10 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 [9–11]. 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 2.3.3 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 2.2.10.

2.4.1 Underlying Type System

We define the underlying dependent type system by a slightly modified version of a fragment of EMLTT [9–11]. 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.

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

Figure 2.1: Some typing rules for the underlying type system. See A.1 for the full definition.

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.

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

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. 2.1

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

Definition 2.4.1 (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 [9, Theorem 4.3.24] and thus gives an example of fibred adjunction models.

Lemma 2.4.2. 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.

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

$$\begin{gathered} \llbracket\Gamma\rrbracket \in \mathbb{B} \qquad \llbracket\Gamma; A\rrbracket \in \mathbb{E}_{\llbracket\Gamma\rrbracket} \qquad \llbracket\Gamma; \underline{C}\rrbracket \in \mathbb{C}_{\llbracket\Gamma\rrbracket} \\ \llbracket\Gamma; V\rrbracket \in \mathbb{E}_{\llbracket\Gamma\rrbracket}(1\llbracket\Gamma\rrbracket, A) \qquad \text{for some } A \\ \llbracket\Gamma; M\rrbracket \in \mathbb{E}_{\llbracket\Gamma\rrbracket}(1\llbracket\Gamma\rrbracket, UC) \qquad \text{for some } C \in \mathbb{C} \end{gathered}$$

Most of the definition of $\llbracket - \rrbracket$ are the same as [9]. For base type constructors b and value constants c, we define $\llbracket - \rrbracket$ as follows.

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

Here, left-hand sides are defined if right-hand sides are defined. The full definition of $\llbracket - \rrbracket$ is listed in Appendix A.

Proposition 2.4.3 (Soundness)**.** Assume that $\llbracket b\rrbracket \in \mathbb{E}_{\{\llbracket\diamond; A\rrbracket\}}$ holds for each $b : A \to \mathrm{Type}$ such that $\llbracket\diamond; A\rrbracket$ is defined, and $\llbracket c\rrbracket \in \mathbb{E}_1(1, \llbracket\diamond; \mathrm{ty}(c)\rrbracket)$ holds if $\llbracket\diamond; \mathrm{ty}(c)\rrbracket \in \mathbb{E}_1$ is defined. Interpretations $\llbracket - \rrbracket$ 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.

Proof. See [9]. □

2.4.2 Predicate Logic

We define syntax for logical formulas by

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

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 [86]), 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 [9, 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. Rules for well-formedness are shown in Fig. 2.2

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

$$\begin{aligned} \llbracket\Gamma \vdash \top\rrbracket &= \top\llbracket\Gamma\rrbracket \\ \llbracket\Gamma \vdash p \wedge q\rrbracket &= \llbracket\Gamma \vdash p\rrbracket \wedge \llbracket\Gamma \vdash q\rrbracket \\ \llbracket\Gamma \vdash p \Rightarrow q\rrbracket &= \llbracket\Gamma \vdash p\rrbracket \Rightarrow \llbracket\Gamma \vdash q\rrbracket \\ \llbracket\Gamma \vdash \forall x : A.p\rrbracket &= \prod_{\llbracket\Gamma; A\rrbracket} \llbracket\Gamma, x : A \vdash p\rrbracket \\ \llbracket\Gamma \vdash V =_A W\rrbracket &= (s\llbracket\Gamma; V\rrbracket)^*(s(\pi^*_{\llbracket\Gamma; A\rrbracket}\llbracket\Gamma; W\rrbracket))^*\mathrm{Eq}(\top\{\llbracket\Gamma; A\rrbracket\}) \\ \llbracket\Gamma \vdash a(V)\rrbracket &= s(\llbracket\Gamma; V\rrbracket)^*\{\overline{!_{\llbracket\Gamma\rrbracket}}(\llbracket\diamond; A\rrbracket)\}^*\llbracket a\rrbracket \end{aligned}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash \top : \mathrm{Prop}} \qquad \frac{\Gamma \vdash p : \mathrm{Prop} \qquad \Gamma \vdash q : \mathrm{Prop}}{\Gamma \vdash p \land q : \mathrm{Prop}}$$

$$\frac{\Gamma \vdash p : \mathrm{Prop} \qquad \Gamma \vdash q : \mathrm{Prop}}{\Gamma \vdash p \Rightarrow q : \mathrm{Prop}} \qquad \frac{\Gamma, x : A \vdash p : \mathrm{Prop}}{\Gamma \vdash \forall x : A.p : \mathrm{Prop}}$$

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

Figure 2.2: Rules for well-formed predicates.

where $\top : \mathbb{B} \to \mathbb{P}$ is the terminal object functor, $a : A \to \mathrm{Prop}$ is a predicate symbol, $s : \mathbb{E}_I(1I, X) \cong \{f : I \to \{X\} \mid \pi_X \circ f = \mathrm{id}_I\}$ is the bijection defined in Section 2.1 for each $I \in \mathbb{B}$ and $X \in \mathbb{E}_I$, and $\mathrm{Eq} : \mathbb{P}_{[\![\Gamma;A]\!]} \to \mathbb{P}_{\pi^*_{[\![\Gamma;A]\!]}[\![\Gamma;A]\!]}$ is a left adjoint of $\delta^*_{[\![\Gamma;A]\!]}$.

2.4.3 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.

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

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 $|\Gamma, x : A| = |\Gamma|, x : |A|$.

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. 2.3. 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

$$\frac{\begin{array}{c} b : A_u \to \mathrm{Type} \qquad \vdash \Gamma \qquad |\Gamma| \vdash b_{A_u}(V) \\ |\Gamma|, v : b_{A_u}(V) \vdash p : \mathrm{Prop} \end{array}}{\Gamma \vdash \{v : b_{A_u}(V) \mid p\}}$$

$$\frac{\begin{array}{c} \vdash \Gamma \qquad |\Gamma| \vdash b_{A_u}(V) = b_{A_u}(W) \\ \Gamma; v : b_{A_u}(V) \mid p \vdash q \end{array}}{\Gamma \vdash \{v : b_{A_u}(V) \mid p\} <: \{v : b_{A_u}(W) \mid q\}}$$

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

$$\frac{\begin{array}{c} \Gamma \vdash A_2 <: A_1 \\ \Gamma, x : A_1 \vdash \underline{C}_1 \qquad \Gamma, x : A_2 \vdash \underline{C}_1 <: \underline{C}_2 \end{array}}{\Gamma \vdash \Pi x{:}A_1.\underline{C}_1 <: \Pi x{:}A_2.\underline{C}_2} \qquad \frac{\begin{array}{c} \Gamma_2 \vdash V \ : \ A \\ \vdash \Gamma_1 <: \Gamma_2 \qquad \Gamma_1 \vdash A <: B \end{array}}{\Gamma_1 \vdash V \ : \ B}$$

$$\frac{\vdash \Gamma}{\Gamma \vdash * \ : \ \{v : 1 \mid \top\}} \qquad \frac{\vdash \Gamma \qquad |\Gamma|, v : 1 \vdash p : \mathrm{Prop}}{\Gamma \vdash \{v : 1 \mid p\}}$$

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

Figure 2.3: Some typing rules for the refinement type system. See A.2 for the full definition.

rules for the judgement $\Gamma; v : A_u \mid p \vdash q$ but assume soundness of the judgement (explained later). We allow "selfification" [86] 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 2.4.4. 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 2.4.5. 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 $\mathrm{int} : 1 \to \mathrm{Type}$ for integers and a value constant for comparison.

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

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

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

where $(\leq') = \text{\textbf{force}}\ (\leq)$. 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.

2.4.4 Semantics

Definition 2.4.6 (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.

$$\begin{tikzcd}
\mathbb{E} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "p"'] & & \mathbb{C} \arrow[ll, bend left, "U"] \arrow[dl] \\
& \mathbb{B} &
\end{tikzcd}
\quad
\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[rr, bend left, "\dot{F}"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\{p|q\}"'] & & \mathbb{D} \arrow[ll, bend left, "\dot{U}"] \arrow[dl] \\
& \mathbb{P} &
\end{tikzcd}
\quad
\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[r, "u"] \arrow[d, "\{p|q\}"] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{P} \arrow[r, "q"] & \mathbb{B}
\end{tikzcd}
\quad
\begin{tikzcd}
\mathbb{D} \arrow[r, "v"] \arrow[d] & \mathbb{C} \arrow[d] \\
\mathbb{P} \arrow[r, "q"] & \mathbb{B}
\end{tikzcd}
\tag{2.7}$$

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 2.2.10 and Lemma 2.2.15. The underlying fibred adjunction model $F \dashv U$ is used for the underlying type system in Section 2.4.1, and $q : \mathbb{P} \to \mathbb{B}$ is for predicate logic in Section 2.4.2. One way to obtain such liftings of fibred adjunction models is to apply the Eilenberg-Moore construction to the monad morphism in Theorem 2.3.3, 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 $[\![\Gamma]\!] \in \mathbb{P}$, $[\![\Gamma; A]\!] \in \{\mathbb{E} \mid \mathbb{P}\}_{[\![\Gamma]\!]}$, and $[\![\Gamma; \underline{C}]\!] \in \mathbb{D}_{[\![\Gamma]\!]}$ similarly to the underlying type system but with the following modification. Here, we make use of the definition of $\{\mathbb{E} \mid \mathbb{P}\}$.

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

See A.2 for the full definition of $[\![-]\!]$.

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 2.4.7.

- If $[\![\Gamma]\!]$ is defined, then $[\![|\Gamma|]\!]$ is defined and equal to $q[\![\Gamma]\!]$.

- If $\llbracket\Gamma; A\rrbracket$ is defined, then $\llbracket|\Gamma|; |A|\rrbracket$ is defined and equal to $u\llbracket\Gamma; A\rrbracket$.
- If $\llbracket\Gamma; \underline{C}\rrbracket$ is defined, then $\llbracket|\Gamma|; |\underline{C}|\rrbracket$ is defined and equal to $v\llbracket\Gamma; \underline{C}\rrbracket$.

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 the judgement $\Gamma; v : A_u \mid p \vdash q$ in the following sense:

$$\begin{aligned}&\Gamma; v : A_u \mid p \vdash q \quad \text{implies}\\ &\qquad \pi^*_{\llbracket|\Gamma|; A_u\rrbracket}\llbracket\Gamma\rrbracket \wedge \llbracket|\Gamma|, v : A_u \vdash p\rrbracket \leq \llbracket|\Gamma|, v : A_u \vdash q\rrbracket \quad \text{in } \mathbb{P}_{\llbracket|\Gamma|, v:A_u\rrbracket}.\end{aligned} \tag{2.8}$$

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

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

and defining a derivation rule for judgement for logical implication $\Gamma; v : A_u \mid p \vdash q$ by $|\Gamma|, v : A_u \mid \llparenthesis\Gamma\rrparenthesis \wedge p \vdash q$. If the derivation rules for predicate logic $\Gamma_u \mid p \vdash q$ is sound (i.e., $\Gamma_u \mid p \vdash q$ implies $\llbracket\Gamma_u \vdash p\rrbracket \leq \llbracket\Gamma_u \vdash q\rrbracket$), then so are the derivation rule for $\Gamma; v : A_u \mid p \vdash q$. This technique is used in, e.g., [125].

Our proof of soundness follows [9] with a few modifications for the dependent refinement type system. We first define two auxiliary morphisms, semantic projection $\mathrm{proj}_{\Gamma_1;x:A;\Gamma_2} : \llbracket\Gamma_1, x : A, \Gamma_2\rrbracket \to \llbracket\Gamma_1, \Gamma_2\rrbracket$ and semantic substitution $\mathrm{subst}_{\Gamma_1;x:A;\Gamma_2;V} : \llbracket\Gamma_1, \Gamma_2[V/x]\rrbracket \to \llbracket\Gamma_1, x : A, \Gamma_2\rrbracket$ in the base category of $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$. The functor $q : \mathbb{P} \to \mathbb{B}$ maps these morphisms to the corresponding morphisms defined for the underlying type system (see [9] for semantic projection and semantic substitution for the underlying type system).

Definition 2.4.8. Assume $\llbracket\Gamma_1, x : A, \Gamma_2\rrbracket$ and $\llbracket\Gamma_1, \Gamma_2\rrbracket$ are defined. We define $\mathrm{proj}_{\Gamma_1;x:A;\Gamma_2} : \llbracket\Gamma_1, x : A, \Gamma_2\rrbracket \to \llbracket\Gamma_1, \Gamma_2\rrbracket$ inductively by

$$\begin{aligned}\mathrm{proj}_{\Gamma_1;x:A;\diamond} &:= \pi_{\llbracket\Gamma_1;A\rrbracket}\\ \mathrm{proj}_{\Gamma_1;x:A;\Gamma_2,y:B} &:= \{\overline{\mathrm{proj}_{\Gamma_1;x:A;\Gamma_2}(\llbracket\Gamma_1, \Gamma_2; B\rrbracket)}\}\end{aligned}$$

where the latter is defined when $\mathrm{proj}^*_{\Gamma_1;x:A;\Gamma_2}\llbracket\Gamma_1, \Gamma_2; B\rrbracket = \llbracket\Gamma_1, x : A, \Gamma_2; B\rrbracket$.

Lemma 2.4.9. If $\mathrm{proj}_{\Gamma_1;x:A;\Gamma_2} : \llbracket\Gamma_1, x : A, \Gamma_2\rrbracket \to \llbracket\Gamma_1, \Gamma_2\rrbracket$ is defined, then $\mathrm{proj}_{|\Gamma_1|;x:|A|;|\Gamma_2|} : \llbracket|\Gamma_1|, x : |A|, |\Gamma_2|\rrbracket \to \llbracket|\Gamma_1|, |\Gamma_2|\rrbracket$ (for the underlying type system) is defined and equal to $q(\mathrm{proj}_{\Gamma_1;x:A;\Gamma_2})$.

Proof. By induction on the length of Γ_2. □

Lemma 2.4.10 (semantic weakening)**.** Assume $\llbracket\Gamma_1, x : A, \Gamma_2\rrbracket$ and $\llbracket\Gamma_1, \Gamma_2\rrbracket$ are defined.

- $\mathrm{proj}_{\Gamma_1;x:A;\Gamma_2} : \llbracket\Gamma_1, x : A, \Gamma_2\rrbracket \to \llbracket\Gamma_1, \Gamma_2\rrbracket$ is defined.
- If $\llbracket\Gamma_1, \Gamma_2; B\rrbracket$ is defined, then $\llbracket\Gamma_1, x : A, \Gamma_2; B\rrbracket$ is defined and equal to $\mathrm{proj}^*_{\Gamma_1;x:A;\Gamma_2}\llbracket\Gamma_1, \Gamma_2; B\rrbracket$.

- If $\llbracket \Gamma_1, \Gamma_2; \underline{C} \rrbracket$ is defined, then $\llbracket \Gamma_1, x : A, \Gamma_2; \underline{C} \rrbracket$ is defined and equal to $\mathrm{proj}^*_{\Gamma_1; x:A; \Gamma_2} \llbracket \Gamma_1, \Gamma_2; \underline{C} \rrbracket$.

Proof. By induction on the size of the content of $\llbracket - \rrbracket$. The only non-trivial part of the proof is the case of refinement types $\{v : b(V) \mid p\}$.

Assume $\llbracket \Gamma_1, \Gamma_2; \{v : b(V) \mid p\} \rrbracket = (\llbracket |\Gamma_1|, |\Gamma_2| \rrbracket, \llbracket \Gamma_1, \Gamma_2 \rrbracket, \pi^*_{\llbracket |\Gamma_1|, |\Gamma_2| \rrbracket} \llbracket \Gamma_1, \Gamma_2 \rrbracket \wedge \llbracket |\Gamma_1|, |\Gamma_2|, v : b(V) \vdash p \rrbracket)$ is defined. By induction hypothesis, $\mathrm{proj}_{\Gamma_1; x:A; \Gamma_2} : \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket \to \llbracket \Gamma_1, \Gamma_2 \rrbracket$ is defined. By the semantic weakening lemma for the underlying type system and logical formulas, we also have

$$\llbracket |\Gamma_1|, x : |A|, |\Gamma_2|; b(V) \rrbracket = \mathrm{proj}^*_{|\Gamma_1|; x:|A|; |\Gamma_2|} \llbracket |\Gamma_1|, |\Gamma_2|; b(V) \rrbracket$$
$$\llbracket |\Gamma_1|, x : |A|, |\Gamma_2|, v : b(V) \vdash p \rrbracket = \mathrm{proj}^*_{|\Gamma_1|; x:|A|; |\Gamma_2|} \llbracket |\Gamma_1|, |\Gamma_2|, v : b(V) \vdash p \rrbracket.$$

Therefore, $\llbracket \Gamma_1, x : A, \Gamma_2; \{v : b(V) \mid p\} \rrbracket$ is defined. The equation

$$\llbracket \Gamma_1, x : A, \Gamma_2; \{v : b(V) \mid p\} \rrbracket = \mathrm{proj}^*_{\Gamma_1; x:A; \Gamma_2} \llbracket \Gamma_1, \Gamma_2; \{v : b(V) \mid p\} \rrbracket$$

follows by Lemma 2.4.9. □

Definition 2.4.11. If $\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$, $\llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket$ and $\mathrm{subst}_{|\Gamma_1|; x:|A|; |\Gamma_2|; V}$ are defined and $\llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket = \mathrm{subst}^*_{|\Gamma_1|; x:|A|; |\Gamma_2|; V} \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$, then we define $\mathrm{subst}_{\Gamma_1; x:A; \Gamma_2; V} : \llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket \to \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$ by

$$\mathrm{subst}_{\Gamma_1; x:A; \Gamma_2; V} \quad \coloneqq \quad \overline{\mathrm{subst}_{|\Gamma_1|; x:|A|; |\Gamma_2|; V}} (\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket).$$

Lemma 2.4.12 (semantic substitution)**.** Assume $\llbracket \Gamma_1; A \rrbracket$ and $\llbracket |\Gamma_1|; V \rrbracket : 1 \llbracket |\Gamma_1| \rrbracket \to \llbracket |\Gamma_1|; |A| \rrbracket$ are defined, and there exists a lifting $1 \llbracket \Gamma_1 \rrbracket \to \llbracket \Gamma_1; A \rrbracket$ above $\llbracket |\Gamma_1|; V \rrbracket$.

- If $\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$ is defined, then $\llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket$ is defined and equal to $\mathrm{subst}^*_{|\Gamma_1|; x:|A|; |\Gamma_2|; V} \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$, that is, $\mathrm{subst}_{\Gamma_1; x:A; \Gamma_2; V}$ is defined.
- If $\llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket$ is defined, then $\llbracket \Gamma_1, \Gamma_2[V/x]; B[V/x] \rrbracket$ is defined and equal to $\mathrm{subst}^*_{\Gamma_1; x:A; \Gamma_2; V} \llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket$.
- If $\llbracket \Gamma_1, x : A, \Gamma_2; \underline{C} \rrbracket$ is defined, then $\llbracket \Gamma_1, \Gamma_2[V/x]; \underline{C}[V/x] \rrbracket$ is defined and equal to $\mathrm{subst}^*_{\Gamma_1; x:A; \Gamma_2; V} \llbracket \Gamma_1, x : A, \Gamma_2; \underline{C} \rrbracket$.

Proof. By induction on the size of the content of $\llbracket - \rrbracket$.

Assume $\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket$ is defined and $\Gamma_2 = \diamond$. Obviously $\llbracket \Gamma_1 \rrbracket$ is defined. Since $\llbracket |\Gamma_1|; V \rrbracket$ has a lifting $1 \llbracket \Gamma_1 \rrbracket \to \llbracket \Gamma_1; A \rrbracket$, $\mathrm{subst}_{|\Gamma_1|; x:|A|; \diamond; V} = s \llbracket |\Gamma_1|; V \rrbracket$ has a lifting $\llbracket \Gamma_1 \rrbracket \to \llbracket \Gamma_1, x : A \rrbracket$. Therefore, we have

$$\llbracket \Gamma_1 \rrbracket \le (s \llbracket |\Gamma_1|; V \rrbracket)^* \llbracket \Gamma_1, x : A \rrbracket \le (s \llbracket |\Gamma_1|; V \rrbracket)^* \pi^*_{\llbracket \Gamma_1; A \rrbracket} \llbracket \Gamma_1 \rrbracket = \llbracket \Gamma_1 \rrbracket.$$

The case for refinement types $\{v : b(V) \mid p\}$ follows from induction hypothesis and the semantic substitution for the underlying type system and logical formulas.

Other cases are easy. Note that the following square is a pullback because subst is Cartesian and the image of the square by $q : \mathbb{P} \to \mathbb{B}$ is a pullback.

$$\begin{tikzcd}
\llbracket \Gamma_1, \Gamma_2[V/x], y : B[V/x] \rrbracket \arrow[r, "\mathrm{subst}"] \arrow[d, "\pi"] & \llbracket \Gamma_1, x : A, \Gamma_2, y : B \rrbracket \arrow[d, "\pi"] \\
\llbracket \Gamma_1, \Gamma_2[V/x] \rrbracket \arrow[r, "\mathrm{subst}"] & \llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket
\end{tikzcd}$$

□

Lemma 2.4.13 (semantic subsumption)**.** Let Γ_1 and Γ_2 be contexts such that $\vdash |\Gamma_1| = |\Gamma_2|$. Assume $\llbracket \Gamma_1 \rrbracket$ and $\llbracket \Gamma_2 \rrbracket$ are defined, and $\llbracket \Gamma_1 \rrbracket \leq \llbracket \Gamma_2 \rrbracket$ in $\mathbb{P}_{\llbracket |\Gamma_1| \rrbracket}$.

- If $\llbracket \Gamma_2; A \rrbracket$ is defined, then $\llbracket \Gamma_1; A \rrbracket$ is defined and equal to $(\llbracket \Gamma_1 \rrbracket \leq \llbracket \Gamma_2 \rrbracket)^* \llbracket \Gamma_2; A \rrbracket$.
- If $\llbracket \Gamma_2; \underline{C} \rrbracket$ is defined, then $\llbracket \Gamma_1; \underline{C} \rrbracket$ is defined and equal to $(\llbracket \Gamma_1 \rrbracket \leq \llbracket \Gamma_2 \rrbracket)^* \llbracket \Gamma_2; \underline{C} \rrbracket$.

Proof. By induction.

For the case of refinement types, we use the semantic conversion for the underlying type system and logical formulas. Other cases are easy. Note that if $\llbracket \Gamma_1 \rrbracket \leq \llbracket \Gamma_2 \rrbracket$ holds, and $\llbracket \Gamma_1; A \rrbracket$ and $\llbracket \Gamma_2; A \rrbracket$ are defined, then $\llbracket \Gamma_1, x : A \rrbracket \leq \llbracket \Gamma_2, x : A \rrbracket$ holds because $\{\llbracket \Gamma_1 \rrbracket \leq \llbracket \Gamma_2 \rrbracket (\llbracket \Gamma_2; A \rrbracket)\} : \llbracket \Gamma_1, x : A \rrbracket \to \llbracket \Gamma_2, x : A \rrbracket$ is vertical. □

Theorem 2.4.14 (Soundness)**.** Assume

- $\Gamma; v : A_u \mid p \vdash q$ is sound in the sense of (2.8),
- $\llbracket b \rrbracket \in \mathbb{E}_{\{\llbracket \diamond; A \rrbracket\}}$ holds for each $b : A \to \mathrm{Type}$ if $\llbracket \diamond; A \rrbracket$ is defined, and
- $\llbracket c \rrbracket \in \{\mathbb{E} \mid \mathbb{P}\}_1 (1, \llbracket \diamond; \mathrm{ty}(c) \rrbracket)$ holds if $\llbracket \diamond; \mathrm{ty}(c) \rrbracket \in \{\mathbb{E} \mid \mathbb{P}\}_1$ is defined.

Then we have the following.

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

Proof. By induction on type derivation. We can basically construct a lifting of term interpretations in the same way as the interpretation in the underlying type system. Most cases are easy or a straightforward extension of the proof for the underlying type system. Non-trivial cases are the selfification and subtyping rule for computational Π-types.

Selfification. For selfification, assume $\llbracket \Gamma; \{v : b(V) \mid p\} \rrbracket$ is defined and let $(X, P, Q) = \llbracket \Gamma; \{v : b(V) \mid p\} \rrbracket$. Then, we have

$$\begin{aligned} &1\llbracket \Gamma, x : \{v : b(V) \mid p\} \rrbracket = (1\{X\}, Q, \pi^*_{1\{X\}} Q) \\ &\llbracket \Gamma, x : \{v : b(V) \mid p\}; \{v : b(V) \mid x = v\} \rrbracket \\ &= (\pi^*_X X, Q, \pi^*_{\pi^*_X X} Q \wedge h^* \mathrm{Eq}(\top\{\pi^*_{\pi^*_X X} \pi^*_X X\})) \end{aligned}$$

where $h = s(\pi^* \llbracket |\Gamma|, x : b(V), v : b(V); v \rrbracket) \circ s(\llbracket |\Gamma|, x : b(V), v : b(V); x \rrbracket)$. We prove that $\llbracket |\Gamma|, x : b(V); x \rrbracket : 1\{X\} \to \pi^*_X X$ has a lifting $1\llbracket \Gamma, x : \{v : b(V) \mid p\} \rrbracket \to \llbracket \Gamma, x : \{v : b(V) \mid p\}; \{v : b(V) \mid x = v\} \rrbracket$. It suffices to prove

$$\pi^*_{1\{X\}} Q \leq \{\llbracket |\Gamma|, x : b(V); x \rrbracket\}^* (\pi^*_{\pi^*_X X} Q \wedge h^* \mathrm{Eq}(\top\{\pi^*_{\pi^*_X X} \pi^*_X X\})).$$

Since $\pi_{\pi^*_X X} \circ \{[\![|\Gamma|, x : b(V); x]\!]\} = \pi_{1\{X\}}$, we can further simplify this to

$$\pi^*_{1\{X\}} Q \leq \{[\![|\Gamma|, x : b(V); x]\!]\}^* h^* \mathrm{Eq}(\top\{\pi^*_{\pi^*_X X} \pi^*_X X\}).$$

First note the following equation.

$$\begin{aligned} h \circ \delta_X &= s(\pi^*[\![|\Gamma|, x : b(V), v : b(V); v]\!]) \circ s([\![|\Gamma|, x : b(V), v : b(V); x]\!]) \circ \delta_X \\ &= s(\pi^*[\![|\Gamma|, x : b(V), v : b(V); v]\!]) \circ \{\overline{\delta_X}(\ldots)\} \circ s(\delta^*_X \pi^*[\![|\Gamma|, x : b(V); x]\!]) \\ &= s(\pi^*[\![|\Gamma|, x : b(V), v : b(V); v]\!]) \circ \delta \circ \delta \\ &= \{\overline{\delta}(\ldots)\} \circ s(\delta^* \pi^*[\![|\Gamma|, x : b(V), v : b(V); v]\!]) \circ \delta \\ &= \delta \circ \delta \circ \delta \end{aligned}$$

So, we get the following inequality by considering a mate of adjunction $\mathrm{Eq} \dashv \delta^*$.

$$h^* \mathrm{Eq}(\top\{\pi^*_{\pi^*_X X} \pi^*_X X\}) \geq \mathrm{Eq}((\delta \circ \delta)^* \top\{\pi^*_{\pi^*_X X} \pi^*_X X\}) = \mathrm{Eq}(\top\{X\})$$

We also have $\pi^*_{1\{X\}} Q \leq \{[\![|\Gamma|, x : b(V); x]\!]\}^* \mathrm{Eq}(Q)$ by the following composite of morphisms.

$$\begin{tikzcd}
\pi^*_{1\{X\}} Q \arrow[d, "\pi^*_{1\{X\}} \eta^{\mathrm{Eq} \dashv \delta^*}"] \\
\pi^*_{1\{X\}} \delta^*_X \mathrm{Eq}(Q) \arrow[d, "\overline{\pi_{1\{X\}}}(\delta^*_X \mathrm{Eq}(Q))"] \\
\delta^*_X \mathrm{Eq}(Q) \arrow[d, equal] \\
(\eta^{1 \dashv \{-\}}_{\{X\}})^* \{[\![|\Gamma|, x : b(V); x]\!]\}^* \mathrm{Eq}(Q) \arrow[d, "\overline{\eta^{1 \dashv \{-\}}_{\{X\}}}(\ldots)"] \\
\{[\![|\Gamma|, x : b(V); x]\!]\}^* \mathrm{Eq}(Q)
\end{tikzcd}$$

Note that $\eta^{1 \dashv \{-\}}_{\{X\}}$ is isomorphic and $\pi_{1\{X\}}$ is the inverse of $\eta^{1 \dashv \{-\}}_{\{X\}}$, so this composite is vertical.

Therefore, we have the following inequality.

$$\begin{aligned} \pi^*_{1\{X\}} Q &\leq \{[\![|\Gamma|, x : b(V); x]\!]\}^* \mathrm{Eq}(Q) \\ &\leq \{[\![|\Gamma|, x : b(V); x]\!]\}^* \mathrm{Eq}(\top\{X\}) \\ &\leq \{[\![|\Gamma|, x : b(V); x]\!]\}^* h^* \mathrm{Eq}(\top\{\pi^*_{\pi^*_X X} \pi^*_X X\}) \end{aligned}$$

Subtyping for computational Π-types. For the subtyping rule of computational Π-types, consider the following composite.

$$\begin{tikzcd}
\dot{U}\prod_{\llbracket\Gamma;A_1\rrbracket}\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket \arrow[d, "\zeta_{\llbracket\Gamma_1;A\rrbracket}"] \\
\prod_{\llbracket\Gamma;A_1\rrbracket}\dot{U}\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket \arrow[d, "\eta"] \\
\prod_{\llbracket\Gamma;A_2\rrbracket}\pi^*_{\llbracket\Gamma;A_2\rrbracket}\prod_{\llbracket\Gamma;A_1\rrbracket}\dot{U}\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket \arrow[d, equal] \\
\prod_{\llbracket\Gamma;A_2\rrbracket}(\llbracket\Gamma;A_2\rrbracket <: \llbracket\Gamma;A_1\rrbracket)^*\pi^*_{\llbracket\Gamma;A_1\rrbracket}\prod_{\llbracket\Gamma;A_1\rrbracket}\dot{U}\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket \arrow[d, "\prod_{\llbracket\Gamma;A_2\rrbracket}(\llbracket\Gamma;A_2\rrbracket<:\llbracket\Gamma;A_1\rrbracket)^*\epsilon"] \\
\prod_{\llbracket\Gamma;A_2\rrbracket}(\llbracket\Gamma;A_2\rrbracket <: \llbracket\Gamma;A_1\rrbracket)^*\dot{U}\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket \arrow[d, equal] \\
\prod_{\llbracket\Gamma;A_2\rrbracket}\dot{U}(\llbracket\Gamma;A_2\rrbracket <: \llbracket\Gamma;A_1\rrbracket)^*\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket \arrow[d, equal] \\
\prod_{\llbracket\Gamma;A_2\rrbracket}\dot{U}\llbracket\Gamma, x : A_2; \underline{C}_1\rrbracket \arrow[d, "\prod_{\llbracket\Gamma;A_2\rrbracket}(\dot{U}\llbracket\Gamma,x:A_2;\underline{C}_1\rrbracket<:\dot{U}\llbracket\Gamma,x:A_2;\underline{C}_2\rrbracket)"] \\
\prod_{\llbracket\Gamma;A_2\rrbracket}\dot{U}\llbracket\Gamma, x : A_2; \underline{C}_2\rrbracket \arrow[d, "\zeta^{-1}_{\llbracket\Gamma_2;A\rrbracket}"] \\
\dot{U}\prod_{\llbracket\Gamma;A_2\rrbracket}\llbracket\Gamma, x : A_2; \underline{C}_2\rrbracket
\end{tikzcd}$$

Note that the image of the composite by $u : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{E}$ is an identity, and both domain and codomain of the composite are in $\{\mathbb{E} \mid \mathbb{P}\}_{\llbracket\Gamma\rrbracket}$. Therefore this composite is the subtyping $\dot{U}\prod_{\llbracket\Gamma;A_1\rrbracket}\llbracket\Gamma, x : A_1; \underline{C}_1\rrbracket <: \dot{U}\prod_{\llbracket\Gamma;A_2\rrbracket}\llbracket\Gamma, x : A_2; \underline{C}_2\rrbracket$. □

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

Corollary 2.4.16. Assume the lifting of fibred adjunction models is given by applying the Eilenberg-Moore construction to a lifting of monads in Theorem 2.3.3. If $\Gamma \vdash M : FA$, then $\theta \circ s\llbracket|\Gamma|; M\rrbracket : \llbracket|\Gamma|\rrbracket \to T\{\llbracket|\Gamma|; A\rrbracket\}$ has a lifting of type $\llbracket\Gamma\rrbracket \to \dot{T}\{\llbracket\Gamma; A\rrbracket\}$ along $q : \mathbb{P} \to \mathbb{B}$. □

2.5 Toward Recursion in Refinement Type Systems

We consider how to deal with general recursion in dependent refinement type systems. In [11], 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 [106] 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) **CFam**(**CPO**) → **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.

2.5.1 Conway Operators

The notion of Conway operators for cartesian categories is defined in [106]. 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 2.5.1 (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 Section 2.1.

Lemma 2.5.2. 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 $\mathbf{s}(\mathbb{B})$.

Proof. Given a Conway operator $(-)^{\dagger}$ on $\mathbb{B}$, we define $(-)^{\ddagger}$ on $s(\mathbb{B}) \to \mathbb{B}^{\to}$ by

$$(\mathrm{id}_I, f)^{\ddagger} = (\mathrm{id}_I, f^{\dagger} \circ \pi_1) : (I, 1) \to (I, X) \tag{2.9}$$

for each $f \in s(\mathbb{B})_I((I, X), (I, X))$.

Given a Conway operator $(-)^{\ddagger}$ on $s(\mathbb{B}) \to \mathbb{B}^{\to}$, we define $(-)^{\dagger}$ by

$$f^{\dagger} = f' \circ \langle \mathrm{id}_I, ! \rangle$$

where $(\mathrm{id}_I, f') = (\mathrm{id}_I, f)^{\ddagger}$ for each $f : I \times X \to X$.

It is easy to verify that these constructions are mutually inverse. Moreover, this is a bijection between Conway operators on $\mathbb{B}$ and Conway operators on $s(\mathbb{B}) \to \mathbb{B}$ because we have the following equations if we assume (2.9).

Dinaturality. For each $f : I \times X \to Y$ and $g : I \times Y \to X$, we have the following equations.

$$((\mathrm{id}_I, g) \circ (\mathrm{id}_I, f))^{\ddagger} = (\mathrm{id}_I, (g \circ \langle \pi_1, f \rangle)^{\dagger} \circ \pi_1)$$
$$(\mathrm{id}_I, g) \circ ((\mathrm{id}_I, f) \circ (\mathrm{id}_I, g))^{\ddagger} = (\mathrm{id}_I, g \circ \langle \mathrm{id}_I, (f \circ \langle \pi_1, g \rangle)^{\dagger} \rangle \circ \pi_1)$$

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

$$\frac{\Gamma \vdash \underline{C} \qquad \Gamma, x : U\underline{C} \vdash M \;:\; \underline{C}}{\Gamma \vdash M[\mathbf{thunk}\ (\mu x : U\underline{C}.M)/x] = \mu x : U\underline{C}.M \;:\; \underline{C}}$$

$$\frac{\Gamma \vdash \underline{C} \qquad \Gamma, x : U\underline{C}, y : U\underline{C} \vdash M \;:\; \underline{C}}{\Gamma \vdash \mu x : U\underline{C}.\mu y : U\underline{C}.M = \mu x : U\underline{C}.M[x/y] \;:\; \underline{C}}$$

Figure 2.4: Typing rules for general recursion.

Naturality. For each $f : I \times X \to X$ and $u : J \to I$, we have the following equations.

$$u^*(\mathrm{id}_I, f)^\ddagger = (\mathrm{id}_I, f^\dagger \circ u \circ \pi_1)$$
$$(u^*(\mathrm{id}_I, f))^\ddagger = (\mathrm{id}_I, (f \circ (u \times \mathrm{id}_X))^\dagger \circ \pi_1)$$

Diagonal property. For each $f : (I \times X) \times X \to X$, we have

$$(\phi((\mathrm{id}_{I\times X}, f)^\ddagger))^\ddagger = (\mathrm{id}_I, (f^\dagger)^\dagger \circ \pi_1)$$
$$(\phi(\delta^*_{(I,X)}(\phi^{-1}(f))))^\ddagger = (\mathrm{id}_I, (f \circ (\mathrm{id}_I \times \Delta_X))^\dagger \circ \pi_1)$$

where $\Delta_X : X \to X \times X$ is the diagonal morphism in $\mathbb{B}$. □

Example 2.5.3. 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 [11].

2.5.2 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. 2.4.

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 2.5.4. 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 2.5.5. Soundness (Proposition 2.4.3) 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 [9, Proposition 4.1.14]. $\square$

2.5.3 Recursion in Refinement Type System

Syntax. We add the typing rule for $\Gamma \vdash \mu x{:}U\underline{C}.M \ : \ \underline{C}$ in Fig. 2.4 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 2.5.6. 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 2.5.7. Let (u, v) be a morphism of CCompCs defined in Theorem 2.2.10. 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 $q : \mathbb{P} \to \mathbb{B}$. $\square$

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 $\{f^{\ddagger}\}$. $\square$

We assume that a lifting of fibred adjunction models (2.7) together with a lifting of Conway operators defined on computation types is given.

Theorem 2.5.8. Soundness (Theorem 2.4.14) holds for the refinement type system extended with general recursion. $\square$

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 2.2.10 because $\mathbf{Adm}(\mathbf{CPO}) \to \mathbf{CPO}$ is not a fibred ccc [49, §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.

2.6 Related Work

Dependent refinement types. Historically, there are two kinds of refinement types. One is *datasort refinement types* [41], which are subsets of underlying types but not necessarily dependent. The other is *index refinement* *types* [126]. 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 [39, 97], which we call *dependent refinement types* or just *refinement types*. Dependent refinement types are widely studied [14, 67, 76, 116], and implemented in, e.g., F* [110, 111] and LiquidHaskell [94, 124, 125]. 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 [116] contains types for computations indexed by quantifiers Q_1Q_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_1Q_2 corresponds to four cartesian liftings of the monad $P_+((-) + 1)$ [8, 66]. 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 [14]. Their system seems to use a graded lifting of the distribution monad where the lifting is graded by privacy parameters, as pointed out in [101]. We leave for future work combining our refinement type system with effect systems based on graded monads [42, 65, 82].

Categorical semantics. Our interpretation of refinement type systems is based on a morphism of CCompCs, which is a similar strategy to [83]. The difference is that our thesis 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 [9–11]. 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 [53, 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 [53] 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 $\mathbb{P}$.

2.7 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 [42, 65, 82]. We hope that this extension will give us a more expressive framework that subsumes, for example, dependent refinement type systems in [14, 116]. Another direction is to define interpretations of $\{v : A_u \mid p\}$ in the style of subset types in [53, 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 chapter.

Chapter 3

A Program Logic for Effect Handlers

In this chapter, we provide a compositional program logic for effect handlers based on operation-wise liftings of Eilenberg-Moore algebras.

Workflow of verification. The usage of our program logic is illustrated as follows (see Table 3.1).

1. Define a set of operations and equations. We define operations that cause computational effects and equations between terms. We get a strong monad T on $\mathbf{Set}$ that represents the computational effects by the correspondence between algebraic presentations and monads [90].

2. Write a program using algebraic operations and effect handlers. We can define implementations of algebraic operations as an effect handler [92].

3. Define a set of truth values. We define a poset Ω as a set of truth values. For example, a classical choice of Ω is two-element boolean algebra $\{\bot, \top\}$. For quantitative property like expectation of probabilistic programs, we can also use $[0, \infty]$ as Ω. The poset Ω induces a fibration $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$ [8] where each object in $\mathbf{Set}/\Omega$ represents an Ω-valued predicate $P : X \to \Omega$.

4. Define a monotone weakest precondition transformer for each operation. This induces a monotone Eilenberg-Moore T-algebra $o : T\Omega \to \Omega$ (see Section 3.2) and further induces a lifting $\dot{T}$ of the monad T. This lifting gives a mapping from a predicate $P : X \to \Omega$ on values to a predicate $\dot{T}P : TX \to \Omega$ on computations [8].

5. Provide a specification and apply our program logic (Section 3.6).

3.1 Preliminaries

We review notions required in this chapter and fix notations. For fibrations, see also 2.1.1.

Table 3.1: Ingredients for our program logic for effect handlers and corresponding notions in categorical semantics.

For programmers	For category theorists	Details
operations and equations	monad T	[89–91]
effect handlers	T-algebra	[92]
truth values for specification	pointwise ordered object Ω	[8]
monotone weakest precondition transformer for each operation	monotone T-algebra on Ω	§3.2
correctness of an effect handler	lifting of a T-algebra along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$	§3.3-3.6

Given a set X, we sometimes identify an element $x \in X$ with a morphism in $\mathbf{Set}(1, X) \cong X$. We use $\lambda x.f$ to denote a function in $\mathbf{Set}$ that takes x as an argument. This notation is the same as the lambda abstraction in the language we consider, but they can be distinguished by the context.

3.1.1 Ordered Object and Lax Slice Category

Let $\mathbb{C}$ be a category. In this chapter, we focus on a fibration $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$ where Ω is a pointwise ordered object defined below. Note that when we discuss interpretations of effect handlers and their liftings in Section 3.6, we focus on the case of $\mathbb{C} = \mathbf{Set}$, but we keep $\mathbb{C}$ as abstract as possible in the other parts of this chapter.

Definition 3.1.1 (ordered object [8, Definition 4.1])**.** An *ordered object* is an object $\Omega \in \mathbb{C}$ together with a partial order $\leq_X$ on $\mathbb{C}(X, \Omega)$ for each $X \in \mathbb{C}$ such that $P \leq_Y Q$ implies $P \circ f \leq_X Q \circ f$ for each $f : X \to Y$ and $P, Q : Y \to \Omega$.

We often omit the subscript X of $\leq_X$ when it is clear from the context.

Definition 3.1.2 (lax slice category [8])**.** Given an ordered object Ω, the *lax slice category* $\mathbb{C}/\Omega$ is a category where

- an object is a morphism $P : X \to \Omega$ in $\mathbb{C}$ for some X, and
- a morphism from $P : X \to \Omega$ to $Q : Y \to \Omega$ is a morphism $f : X \to Y$ in $\mathbb{C}$ such that $P \leq_X Q \circ f$.

We also define the domain functor $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$ by $\mathrm{dom}\ P = X$ and $\mathrm{dom}\ f = f$. It is known that the domain functor $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$ is a posetal fibration.

The intuition of the lax slice category is as follows. An object $P \in (\mathbb{C}/\Omega)_X = \mathbb{C}(X, \Omega)$ represents an Ω-valued predicate on X. A morphism in $\mathbb{C}/\Omega$ is a morphism in $\mathbb{C}$ that preserves predicates.

One possible way to define an ordered object is to define a partial order to $\mathbb{C}(1, \Omega)$ and extend them to the poinwise order in $\mathbb{C}(X, \Omega)$ for each $X \in \mathbb{C}$.

Lemma 3.1.3. Assume that $\mathbb{C}$ is a well-pointed category with a terminal object 1, i.e., $\mathbb{C}(1, -)$ is faithful. If $(\mathbb{C}(1, \Omega), \leq_1)$ is a partially ordered set, then the relation $P \leq_X Q$ on $\mathbb{C}(X, \Omega)$ defined by

$$\forall x : 1 \to X, P \circ x \leq_1 Q \circ x \tag{3.1}$$

is an ordered object.

Proof. It is easy to check that $\leq_X$ is reflexive and transitive. The anti-symmetry follows from that of $\leq_1$ and well-pointedness of $\mathbb{C}$. For each $f : X \to Y$ and $P \leq_Y Q$, it is easy to prove $P \circ f \leq_X Q \circ f$. □

Based on Lemma 3.1.3, we carve out the specific class of ordered objects.

Definition 3.1.4 (pointwise ordered object)**.** We say an ordered object Ω is *pointwise* if $\mathbb{C}$ is well-pointed and (3.1) implies $P \leq_X Q$ for any $P, Q : X \to \Omega$.

Note that pointwise ordered objects are a strict subclass of ordered objects: for any ordered object Ω, $P \leq_X Q$ implies (3.1), but the converse does not always hold.

Example 3.1.5 (an ordered object that is not pointwise)**.** Let $\mathbb{C} = \mathbf{Set}$ and $\Omega = 2 = \{\bot, \top\}$. We define a poset $(\mathbf{Set}(X, 2), \leq_X)$ by

$$\leq_X \quad \coloneqq \quad \{(\lambda x.\bot, \lambda x.\top)\} \cup \{(f, f) \mid f : X \to 2\}$$

for each $X \in \mathbf{Set}$. Then, $(2, \{\leq_X\}_X)$ is an ordered object but not pointwise.

3.1.2 Monad Lifting

Given a monad T on a category $\mathbb{C}$, an *Eilenberg-Moore T-algebra*, or *T-algebra* is an object $X \in \mathbb{C}$ together with a morphism $\alpha : TX \to X$ in $\mathbb{C}$ such that $\alpha \circ \eta_X = \mathrm{id}_X$ (the unit law) and $\alpha \circ T\alpha = \alpha \circ \mu$ (the associative law) hold.

Given a lifting of a monad T (Definition 2.3.1), we can define the weakest precondition transformer and the Hoare triple for an effectful computation whose computational effect is interpreted by T.

Definition 3.1.6 (weakest precondition transformer and Hoare triple [8, Definition 3.3])**.** Let $p : \mathbb{E} \to \mathbb{B}$ be a posetal fibration and $\dot{T}$ be a lifting of a monad T on $\mathbb{B}$. For each $f : X \to TY$, the *weakest precondition transformer* $\mathrm{wp}[f] : \mathbb{E}_Y \to \mathbb{E}_X$ is defined by $\mathrm{wp}[f](P) \coloneqq f^*\dot{T}P$, and the *Hoare triple* $\{P\}f\{Q\}$ is defined by $\{P\}f\{Q\} \coloneqq (P \leq \mathrm{wp}[f](Q))$.

When we consider $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$, the following fact is known.

Definition 3.1.7 (monotone T-algebra [8, Definition 4.3])**.** A T-algebra $o : T\Omega \to \Omega$ is *monotone* if $i \leq_X i'$ implies $o \circ Ti \leq_{TX} o \circ Ti'$.

Proposition 3.1.8 ([8, Corollary 4.5])**.** There is a bijective correspondence between a monotone T-algebra $o : T\Omega \to \Omega$ and a Cartesian lifting of T along $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$. This bijection is given by $o \mapsto T_o$ where $T_o f \coloneqq o \circ Tf$ for each $f \in \mathbb{C}/\Omega$. □

3.1.3 Algebraic Theory

Following [92], we define several concepts of algebraic theories and their models in **Set**.

Signature types $\mathscr{A}, \mathscr{B}$ are defined by

$$\mathscr{A}, \mathscr{B} \coloneqq b \mid \mathbf{1} \mid \mathscr{A} \times \mathscr{B} \mid \sum_{l \in L} \mathscr{A}_l$$

where b ranges over base types and L ranges over finite sets of labels. A *signature* Σ is a set of typed operation symbol $(\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}) \in \Sigma$.

An *effect theory* $\mathcal{T}$ is a set of equations of the form $\Gamma \mid Z \vdash T_1 = T_2$ where $\Gamma = x_1 : \mathscr{A}_1, \dots, x_n : \mathscr{A}_n$ is a context of value variables, $Z = z_1 : \mathscr{B}_1, \dots, z_m : \mathscr{B}_m$ is a context of template variables ($z : \mathscr{B} \in Z$ means that z takes a term of type $\mathscr{B}$ as an argument), and T_1, T_2 are well-formed templates relative to Γ and Z. Here, *templates* T are defined by

$$\begin{aligned} T \coloneqq{} & z(V) \mid \textbf{match } V \textbf{ with } \langle x, y \rangle \mapsto T \\ & \mid \textbf{match } V \textbf{ with} \{l(x_l) \mapsto T_l\}_{l \in L} \mid \mathrm{op}_V(x{:}\mathscr{B}.T) \end{aligned}$$

where V ranges over value terms defined later. The definition of well-formedness is standard, so we omit it (see [92]).

Example 3.1.9 (state with one location [92])**.** We consider global state that stores one value of type **int**. Let $\Sigma = \{\mathrm{set} : \mathbf{int} \twoheadrightarrow \mathbf{1}, \mathrm{get} : \mathbf{1} \twoheadrightarrow \mathbf{int}\}$. The effect theory for the global state is given as follows.

$$\begin{aligned} \diamond \mid z : \mathbf{int} \times \mathbf{int} \vdash \quad & \mathrm{get}(x.\mathrm{get}(x'.z(x, x'))) = \mathrm{get}(x.z(x, x)) \\ x : \mathbf{int} \mid z : \mathbf{int} \vdash \quad & \mathrm{set}_x(\mathrm{get}(x'.z(x'))) = \mathrm{set}_x(z(x)) \\ \diamond \mid z : \mathbf{1} \vdash \quad & \mathrm{get}(x.\mathrm{set}_x(z)) = z \\ x : \mathbf{int}, x' : \mathbf{int} \mid z : \mathbf{1} \vdash \quad & \mathrm{set}_x(\mathrm{set}_{x'}(z)) = \mathrm{set}_{x'}(z) \end{aligned}$$

Suppose that an interpretation $[\![b]\!] \in \mathbf{Set}$ is given for each base type b. The interpretation $[\![-]\!]$ can be extended to signature types using corresponding operations in **Set**.

A *model* of $\mathcal{T}$ is a pair $(X, \{\mathrm{op}^X\}_{\mathrm{op} \in \Sigma})$ where X is a set and $\mathrm{op}^X : [\![\mathscr{A}]\!] \times X^{[\![\mathscr{B}]\!]} \to X$ is an interpretation of each $:\mathscr{A} \twoheadrightarrow \mathscr{B}$ such that $\{\mathrm{op}^X\}_{\mathrm{op} \in \Sigma}$ satisfies all the equations in $\mathcal{T}$. A *morphism* $f : (X, \{\mathrm{op}^X\}_{\mathrm{op} \in \Sigma}) \to (Y, \{\mathrm{op}^Y\}_{\mathrm{op} \in \Sigma})$ of models is a morphism $f : X \to Y$ such that $\mathrm{op}^Y \circ ([\![\mathscr{A}]\!] \times f^{[\![\mathscr{B}]\!]}) = f \circ \mathrm{op}^X$ holds for each $:\mathscr{A} \twoheadrightarrow \mathscr{B}$. We denote the category of models of $\mathcal{T}$ by $\mathbf{Mod}_{\mathcal{T}}$.

Free model functor We have a left adjoint F to the forgetful functor $U : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}$ by [92, Prop. 4.3], which is defined by $FX \coloneqq \mathbf{Term}_\Sigma X/{\sim}$ where $\mathbf{Term}_\Sigma X$ is defined inductively by

- if $x \in X$, then $x \in \mathbf{Term}_\Sigma X$
- if $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$, $a \in [\![\mathscr{A}]\!]$, and $t \in (\mathbf{Term}_\Sigma X)^{[\![\mathscr{B}]\!]}$, then $\mathrm{op}_a(t) \in \mathbf{Term}_\Sigma X$

and $\sim$ is the equivalence relation induced by $\mathcal{T}$ (see [92] for details). For simplicity, we usually identify an equivalence class $[t]_\sim \in FX$ with its representative t.

Note that the counit $\epsilon^{F \dashv U}$ of the free-forgetful adjunction $F \dashv U : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}$ is given as follows:

$$\epsilon^{F \dashv U}_{(X, \{\mathrm{op}^X\})}([t]_\sim) = \epsilon'(t)$$

where $\epsilon' : \mathbf{Term}_\Sigma X \to X$ is defined inductively by

$$\begin{aligned} \epsilon'(x) &= x && \text{if } x \in X \\ \epsilon'(\mathrm{op}_a(t)) &= \mathrm{op}^X(a, \epsilon' \circ t) && \forall \mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}, a \in [\![\mathscr{A}]\!], t \in (\mathbf{Term}_\Sigma X)^{[\![\mathscr{B}]\!]}. \end{aligned}$$

Using the free-forgetful adjunction $F \dashv U : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}$, we obtain a monad $T \coloneqq UF$ on $\mathbf{Set}$ from the effect theory $\mathcal{T}$.

Example 3.1.10. The effect theory in Example 3.1.9 corresponds to the global state monad $T = (- \times \mathbb{Z})^{\mathbb{Z}}$.

Comparison functor K^T Any model in $\mathbf{Mod}_{\mathcal{T}}$ induces an Eilenberg-Moore T-algebra via the *comparison functor* $K^T : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}^T$ defined by

$$K^T(X, \{\mathrm{op}^X\}) = U\epsilon^{F \dashv U}_{(X, \{\mathrm{op}^X\})} : TX \to X.$$

$$\begin{tikzcd} \mathbf{Mod}_{\mathcal{T}} \arrow[rr, "K^T"] \arrow[dr, bend left, "U"] \arrow[dr, phantom, "\dashv"] & & \mathbf{Set}^T \arrow[dl, bend left, "U^T"] \arrow[dl, phantom, "\dashv"] \\ & \mathbf{Set} \arrow[ul, bend left, "F"] \arrow[ur, bend left, "F^T"] & \end{tikzcd}$$

3.1.4 Syntax and Semantics of Effect Handlers

We review syntax and semantics of the language with effect handlers in [92].

Syntax. We define a functional programming language with effect handlers following [92]. We focus on only some parts of the language in [92] relevant to effect handlers.

The language is based on the call-by-push-value [79]. We have *value types* A, B, *computation types* $\underline{C}$, *value contexts* Γ, and *continuation contexts* K.

$$\begin{aligned} A, B &\coloneqq b \mid \mathbf{1} \mid A \times B \mid \sum_{l \in L} A_l \mid U\underline{C} \\ \underline{C} &\coloneqq FA \mid A \to \underline{C} \mid \prod_{l \in L} \underline{C}_l \\ \Gamma &\coloneqq \diamond \mid \Gamma, x : A \\ K &= \diamond \mid K, k : \mathscr{A} \to \underline{C} \end{aligned}$$

Here, note that value types subsume signature types.

Let $\Sigma = \{\mathrm{op}_1 : \mathscr{A}_1 \twoheadrightarrow \mathscr{B}_1, \ldots, \mathrm{op}_n : \mathscr{A}_n \twoheadrightarrow \mathscr{B}_n\}$ be a signature. *Value terms* V, *computation terms* M, N, and effect handlers H are defined as follows.

$$\begin{aligned} V &\coloneqq x \mid \mathbf{thunk}\ M \mid \ldots \\ M, N &\coloneqq M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x{:}A\ \mathbf{in}\ N \mid \mathrm{op}(V; y{:}\mathscr{B}.M) \mid \mathbf{return}\ V \mid \\ &\qquad M\ \mathbf{to}\ x : A\ \mathbf{in}\ N \mid \mathbf{force}\ V \mid k(V) \mid M\ V \mid \lambda x : A.M \mid \ldots \\ H &\coloneqq \{\mathrm{op}_1(x_1; k_1) \mapsto M_{\mathrm{op}_1}, \ldots, \mathrm{op}_n(x_n; k_n) \mapsto M_{\mathrm{op}_n}\} \end{aligned}$$

$$\frac{\Gamma, x : A \mid K, k : B \to \underline{C} \vdash M_{\mathrm{op}} : \underline{C} \quad \text{for each op} : \mathscr{A} \mathrel{-\!\!\triangleright} \mathscr{B}}{\Gamma \mid K \vdash \{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\} : \underline{C}\ \textbf{handler}}$$

$$\frac{\Gamma \mid K \vdash M : FA \qquad \Gamma \mid K \vdash H : \underline{C}\ \textbf{handler} \qquad \Gamma, x : A \mid K \vdash N : \underline{C}}{\Gamma \mid K \vdash M\ \textbf{handled with}\ H\ \textbf{to}\ x{:}A\ \textbf{in}\ N : \underline{C}}$$

Figure 3.1: Typing rules for effect handlers and handling construct.

Here, x ranges over value variables and k ranges over continuation variables. An effect handler is sometimes denoted by $\{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op} \in \Sigma}$.

We have three kinds of typing judgements for value terms, computation terms, and effect handlers.

$$\Gamma \mid K \vdash V : A \qquad \Gamma \mid K \vdash M : \underline{C} \qquad \Gamma \mid K \vdash H : \underline{C}\ \textbf{handler}$$

Typing rules are standard [92], and we listed the typing rules for effect handlers and handling constructs in Fig. 3.1.

Example 3.1.11 (Continuation of Example 3.1.9)**.** Consider the signature $\Sigma = \{\mathrm{set} : \mathbf{int} \mathrel{-\!\!\triangleright} \mathbf{1}, \mathrm{get} : \mathbf{1} \mathrel{-\!\!\triangleright} \mathbf{int}\}$. We can define an effect handler for these operations as follows.

$$H \coloneqq \{\mathrm{set}(x; k) \mapsto \lambda s : \mathbf{int}.k\ ()\ x, \mathrm{get}(x; k) \mapsto \lambda s : \mathbf{int}.k\ s\ s\}$$

The type of this effect handler is $\diamond \mid \diamond \vdash H : (\mathbf{int} \to F\ \mathbf{int})\ \textbf{handler}$.

Semantics. The semantics of the language is also based on [92]. Most of the semantics is defined as the standard CBPV semantics [79]. Here, the adjunction $F \dashv U : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}$ is used as an *adjunction model*, and value types and computation types are interpreted by the corresponding structure of **Set** and $\mathbf{Mod}_{\mathcal{T}}$, respectively.

$$\begin{gathered}
[\![A]\!] \in \mathbf{Set} \qquad [\![\underline{C}]\!] \in \mathbf{Mod}_{\mathcal{T}} \\
[\![x_1 : A_1, \ldots, x_n : A_n]\!] \coloneqq [\![A_1]\!] \times \cdots \times [\![A_n]\!] \\
[\![k_1 : \mathscr{A}_1 \to \underline{C}_1, \ldots, k_n : \mathscr{A}_n \to \underline{C}_n]\!] \coloneqq (U[\![\underline{C}_1]\!])^{[\![\mathscr{A}_1]\!]} \times \cdots \times (U[\![\underline{C}_n]\!])^{[\![\mathscr{A}_n]\!]} \\
[\![\Gamma \mid K \vdash V : A]\!] : [\![\Gamma]\!] \times [\![K]\!] \to [\![A]\!] \qquad [\![\Gamma \mid K \vdash M : \underline{C}]\!] : [\![\Gamma]\!] \times [\![K]\!] \to U[\![\underline{C}]\!]
\end{gathered}$$

We often write $[\![\Gamma \mid K \vdash V : A]\!]$ and $[\![\Gamma \mid K \vdash M : \underline{C}]\!]$ as $[\![V]\!]$ and $[\![M]\!]$, respectively.

The most essential part of the semantics is that an effect handler $\Gamma \mid K \vdash H : \underline{C}$ **handler** is interpreted as a family of models of $\mathcal{T}$ (or equivalently, Eilenberg-Moore algebras) indexed by the context $[\![\Gamma]\!] \times [\![K]\!]$.

For theoretical clarity, we use an equivalent but slightly different formulation of the semantics of effect handlers. We consider the simple fibration $\mathsf{s}_{\mathbb{C}} : \mathbf{s}(\mathbb{C}) \to \mathbb{C}$ and formulate a family of Eilenberg-Moore algebras indexed by the context as an Eilenberg-Moore algebra of a *fibred* monad. This formulation is an instance of [10] that uses the simple fibration and the split fibred monad induced by a strong monad.

Lemma 3.1.12 ([53, Exercise 2.6.10 (ii)])**.** Let $\mathbb{C}$ be a category with finite products. There is a one-to-one correspondence between strong monads on $\mathbb{C}$ and split fibred monads on the simple fibration $s_{\mathbb{C}} : \mathbf{s}(\mathbb{C}) \to \mathbb{C}$.

Proof. Given a strong monad, we define a split fibred monad by $T'(I, X) \coloneqq (I, TX)$. This gives the one-to-one correspondence. □

Consider the Eilenberg-Moore fibration $(s_{\mathbb{C}})^{T'} : (\mathbf{s}(\mathbb{C}))^{T'} \to \mathbb{C}$ of T'. By [53, Exercise 1.7.9 (ii)], a T'-algebra in $(\mathbf{s}(\mathbb{C}))^{T'}$ is a vertical morphism $(\mathrm{id}_I, \alpha) : T'(I, X) \to (I, X)$ in $s(\mathbb{C})$ that satisfies $(\mathrm{id}_I, \alpha) \circ \eta = \mathrm{id}_{(I,X)}$ and $(\mathrm{id}_I, \alpha) \circ \mu = (\mathrm{id}_I, \alpha) \circ T'(\mathrm{id}_I, \alpha)$. By focusing on the second component α, a T'-algebra over I can be rephrased as a morphism $\alpha : I \times TX \to X$ that satisfies the following condition.

Definition 3.1.13 (simply fibred T-algebra)**.** Let T be a strong monad. A *simply fibred T-algebra* on X over I is a morphism $\alpha : I \times TX \to X$ in $\mathbb{C}$ that makes the following diagram commute.

$$\begin{tikzcd}
I \times X \arrow[r, "\mathrm{id}_I \times \eta"] \arrow[dr, "\pi_2"'] & I \times TX \arrow[d, "\alpha"] & I \times T^2X \arrow[r, "\langle \pi_1, T\alpha \circ \mathrm{st} \rangle"] \arrow[d, "\mathrm{id}_I \times \mu"] & I \times TX \arrow[d, "\alpha"] \\
 & X & I \times TX \arrow[r, "\alpha"] & X
\end{tikzcd}$$

Lemma 3.1.14. If T is a monad on **Set**, then $\alpha : I \times TX \to X$ is a simply fibred T-algebra if and only if $\alpha(i, -) : TX \to X$ is a T-algebra for each $i \in I$. □

Next, we extend the comparison functor $K^T : \mathbf{Mod}_{\mathcal{T}} \to \mathbf{Set}^T$ as follows.

A *simply fibred model* of $\mathcal{T}$ over $I \in \mathbf{Set}$ is a set X together with a family of functions $\{\mathrm{op}^X : I \times \llbracket \mathscr{A} \rrbracket \times X^{\llbracket \mathscr{B} \rrbracket} \to X\}_{\mathrm{op} \in \Sigma}$ such that $\{\mathrm{op}^X(i, -, -) : \llbracket \mathscr{A} \rrbracket \times X^{\llbracket \mathscr{B} \rrbracket} \to X\}_{\mathrm{op} \in \Sigma}$ is a model of $\mathcal{T}$ for each $i \in I$.

Given a simply fibred model $(X, \{\mathrm{op}\}_{\mathrm{op} \in \Sigma})$, we define a fibred T-algebra $K_I^T((X, \{\mathrm{op}^X\}_{\mathrm{op} \in \Sigma}))$ by

$$K_I^T((X, \{\mathrm{op}^X\}_{\mathrm{op} \in \Sigma}))(i, -) \coloneqq K^T((X, \{\mathrm{op}^X(i, -, -)\}_{\mathrm{op} \in \Sigma}))$$

where K^T is the comparison functor.

Given an effect handler $\{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op} \in \Sigma}$, the family of interpretations of M_{op} define a simply fibred model (if they satisfy all equations in $\mathcal{T}$), and thus define a simply fibred T-algebra. We use this as the interpretation $\llbracket \{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op} \in \Sigma} \rrbracket$ of the effect handler.

Definition 3.1.15. For $\Gamma \mid K \vdash \mathrm{op}(V; y{:}\mathscr{B}.M) : \underline{C}$ where $\mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}$, its interpretation is defined by

$$\frac{\begin{array}{c}\llbracket V \rrbracket \ : \ \llbracket \Gamma \rrbracket \times \llbracket K \rrbracket \to \llbracket \mathscr{A} \rrbracket \qquad \llbracket M \rrbracket \ : \ \llbracket \Gamma, y : \mathscr{B} \rrbracket \times \llbracket K \rrbracket \to U\llbracket \underline{C} \rrbracket \\ m = \lambda(\gamma, \kappa).\llbracket M \rrbracket((\gamma, -), \kappa) \ : \ \llbracket \Gamma \rrbracket \times \llbracket K \rrbracket \to (U\llbracket \underline{C} \rrbracket)^{\llbracket \mathscr{B} \rrbracket}\end{array}}{\llbracket \mathrm{op}(V; y{:}\mathscr{B}.M) \rrbracket \ \coloneqq \ \mathrm{op}^{\llbracket \underline{C} \rrbracket} \circ \langle \llbracket V \rrbracket, m \rangle}$$

where $\mathrm{op}^{\llbracket \underline{C} \rrbracket} : \llbracket \mathscr{A} \rrbracket \times (U\llbracket \underline{C} \rrbracket)^{\llbracket \mathscr{B} \rrbracket} \to U\llbracket \underline{C} \rrbracket$ is the interpretation of $\mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}$ in $\llbracket \underline{C} \rrbracket \in \mathbf{Mod}_{\mathcal{T}}$. This definition means that if functions in the upper part are defined, then the function in the lower part is defined.

For an effect handler $\Gamma \mid K \vdash \{\mathrm{op}(x;k) \mapsto M_{\mathrm{op}}\}_{\mathrm{op}\in\Sigma} : \underline{C}$ **handler**, its interpretation is a simply fibred T-algebra $[\![\{\mathrm{op}(x;k) \mapsto M_{\mathrm{op}}\}]\!] : ([\![\Gamma]\!] \times [\![K]\!]) \times TU[\![\underline{C}]\!] \to U[\![\underline{C}]\!]$ defined as follows.

$$\frac{\begin{array}{c} [\![M_{\mathrm{op}}]\!] : ([\![\Gamma]\!] \times [\![\mathscr{A}]\!]) \times ([\![K]\!] \times U[\![\mathscr{B} \to \underline{C}]\!]) \to U[\![\underline{C}]\!] \quad \text{for each } \mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B} \\ m_{\mathrm{op}} = \lambda((\gamma,\kappa),a,k).[\![M_{\mathrm{op}}]\!]((\gamma,a),(\kappa,k)) \\ \quad : \quad ([\![\Gamma]\!] \times [\![K]\!]) \times [\![\mathscr{A}]\!] \times U[\![\mathscr{B} \to \underline{C}]\!] \to U[\![\underline{C}]\!] \\ (U[\![\underline{C}]\!], \{m_{\mathrm{op}}\}_{\mathrm{op}\in\Sigma}) \text{ is a simply fibred model of } \mathcal{T} \end{array}}{[\![\{\mathrm{op}(x;k) \mapsto M_{\mathrm{op}}\}]\!] \coloneqq K^T_{[\![\Gamma]\!]\times[\![K]\!]}(U[\![\underline{C}]\!], \{m_{\mathrm{op}}\}_{\mathrm{op}\in\Sigma})}$$

Here, note that we have $U[\![B \to \underline{C}]\!] = (U[\![\underline{C}]\!])^{[\![B]\!]}$ in any adjunction model. Note also that we use m_{op} to change the order of arguments of $[\![M_{\mathrm{op}}]\!]$.

For a handling construct $\Gamma \mid K \vdash M$ **handled with** H **to** $x{:}A$ **in** $N : \underline{C}$, its interpretation is defined as follows.

$$\frac{\begin{array}{c} [\![M]\!] \; : \; [\![\Gamma]\!] \times [\![K]\!] \to UF[\![A]\!] \\ [\![H]\!] \; : \; ([\![\Gamma]\!] \times [\![K]\!]) \times TU[\![\underline{C}]\!] \to U[\![\underline{C}]\!] \text{ is a simply fibred } T\text{-algebra} \\ [\![N]\!] \; : \; ([\![\Gamma]\!] \times [\![A]\!]) \times [\![K]\!] \to U[\![\underline{C}]\!] \\ n = \lambda((\gamma,\kappa),a).[\![N]\!]((\gamma,a),\kappa) \; : \; ([\![\Gamma]\!] \times [\![K]\!]) \times [\![A]\!] \to U[\![\underline{C}]\!] \end{array}}{[\![M \textbf{ handled with } H \textbf{ to } x{:}A \textbf{ in } N]\!] \coloneqq [\![H]\!] \circ \langle \pi_1, UFn \circ \mathrm{st}\rangle \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![M]\!]\rangle}$$

Example 3.1.16 (Continuation of Example 3.1.11)**.** The effect handler H in Example 3.1.11 is interpreted as the simply fibred T-algebra on $U(F\mathbb{Z})^{\mathbb{Z}}$

$$[\![H]\!] = K^T_1(U(F\mathbb{Z})^{\mathbb{Z}}, \{\mathrm{set}^{[\![H]\!]}, \mathrm{get}^{[\![H]\!]}\}) : 1 \times TU(F\mathbb{Z})^{\mathbb{Z}} \to U(F\mathbb{Z})^{\mathbb{Z}}$$

induced by the following interpretations of operations.

$$\begin{aligned} \mathrm{get}^{[\![H]\!]} &= [\![x : \mathbf{1} \mid k : \mathbf{int} \to (\mathbf{int} \to F\ \mathbf{int}) \vdash \lambda s.k\ s\ s]\!] \\ &\quad : 1 \times 1 \times (U(F\mathbb{Z})^{\mathbb{Z}})^{\mathbb{Z}} \to U(F\mathbb{Z})^{\mathbb{Z}} \\ \mathrm{set}^{[\![H]\!]} &= [\![x : \mathbf{int} \mid k : \mathbf{1} \to (\mathbf{int} \to F\ \mathbf{int}) \vdash \lambda s.k\ ()\ x]\!] \\ &\quad : 1 \times \mathbb{Z} \times (U(F\mathbb{Z})^{\mathbb{Z}})^1 \to U(F\mathbb{Z})^{\mathbb{Z}} \end{aligned}$$

3.2 Weakest Preconditions for Algebraic Operations

In this section, we discuss the relationship between a Cartesian lifting of a monad and monotone weakest precondition transformers of each operation $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$.

Let T be the monad that corresponds to the effect theory $\mathcal{T}$ and Ω be an ordered object in **Set**. By Proposition 3.1.8, a Cartesian lifting of T along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$ corresponds to a monotone T-algebra $o : T\Omega \to \Omega$. We consider defining a monotone T-algebra $o : T\Omega \to \Omega$ by giving an interpretation $\mathrm{op}^{\Omega} : [\![\mathscr{A}]\!] \times \Omega^{[\![\mathscr{B}]\!]} \to \Omega$ of each operation $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$. We show that the monotonicity of o is equivalent to the monotonicity of $\mathrm{op}^{\Omega} : [\![\mathscr{A}]\!] \times \Omega^{[\![\mathscr{B}]\!]} \to \Omega$.

Theorem 3.2.1 (operation-wise condition for monotone T-algebra)**.** Assume Ω is pointwise. Let $(\Omega, \{\mathrm{op}^{\Omega}\}_{\mathrm{op}}) \in \mathbf{Mod}_{\mathcal{T}}$ be a model. The T-algebra $K^T(\Omega, \{\mathrm{op}^{\Omega}\}_{\mathrm{op}})$ is monotone if and only if each op^{Ω} is monotone, that is, $P \leq_{[\![\mathscr{B}]\!]} Q$ implies $\mathrm{op}^{\Omega}(-,P) \leq_{[\![\mathscr{A}]\!]} \mathrm{op}^{\Omega}(-,Q)$ for any $P, Q : [\![\mathscr{B}]\!] \to \Omega$.

Proof. Let $o \coloneqq K^T(\Omega, \{\mathrm{op}^\Omega\}_{\mathrm{op}})$. Since Ω is pointwise, the monotonicity of o is equivalent to the condition that $i \leq_X i'$ implies $o \circ Ti \circ x \leq_{TX} o \circ Ti' \circ x$ for any $i, i' \in \mathbf{Set}(X, \Omega)$ and $x \in \mathbf{Set}(1, TX)$.

The "if" part is proved by induction on (a representative of) x.

- If $x = \eta \circ x'$ for some $x' \in \mathbf{Set}(1, X)$,
$$o \circ Ti \circ x = i \circ x' \leq i' \circ x' = o \circ Ti' \circ x.$$
- If $x = \mathrm{op}_a(t)$ for some $t : [\![\mathscr{B}]\!] \to T\Omega$, then we have
$$\begin{aligned} o \circ Ti \circ x &= o \circ \mathrm{op}_a(Ti \circ t) \\ &= \mathrm{op}^\Omega(a, o \circ Ti \circ t) \\ &\leq \mathrm{op}^\Omega(a, o \circ Ti' \circ t) \\ &= o \circ Ti' \circ x \end{aligned}$$
by the induction hypothesis and the monotonicity of op^Ω.

For the "only if" part, assume $P \leq Q$. By the monotonicity of o, we have $o \circ TP \circ \mathrm{op}_a(\eta) \leq o \circ TQ \circ \mathrm{op}_a(\eta)$ for each $a \in [\![\mathscr{A}]\!]$ where $\eta : [\![\mathscr{B}]\!] \to T[\![\mathscr{B}]\!]$. This inequality simplifies to $\mathrm{op}^\Omega(a, P) \leq \mathrm{op}^\Omega(a, Q)$ for each $a \in [\![\mathscr{A}]\!]$. □

Moreover, we can think of $\mathrm{op}^\Omega : [\![\mathscr{A}]\!] \times \Omega^{[\![\mathscr{B}]\!]} \to \Omega$ as the weakest precondition transformer of the generic effect for $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$.

Proposition 3.2.2 (weakest precondition transformer of generic effects)**.** Let $\mathrm{gen}_{\mathrm{op}} : [\![\mathscr{A}]\!] \to T[\![\mathscr{B}]\!]$ be the *generic effect* defined by $\mathrm{gen}_{\mathrm{op}}(a) = \mathrm{op}_a(\eta)$ for each $a \in [\![\mathscr{A}]\!]$. Let $\dot{T}$ be a Cartesian lifting induced by a monotone T-algebra $o \coloneqq K^T(\Omega, \{\mathrm{op}^\Omega\})$. Then, we have
$$\mathrm{wp}[\mathrm{gen}_{\mathrm{op}}](P) = \mathrm{op}^\Omega(-, P)$$
for each $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$ and $P : [\![\mathscr{B}]\!] \to \Omega$.

Proof. By easy calculation: for each $a : 1 \to [\![\mathscr{A}]\!]$,
$$o \circ TP \circ \mathrm{gen}_{\mathrm{op}} \circ a = o \circ TP \circ \mathrm{op}_a(\eta) = \mathrm{op}^\Omega(a, P) = \mathrm{op}^\Omega(-, P) \circ a.$$
□

Therefore, to obtain a Cartesian lifting of T along $\mathrm{dom} : \mathbf{Set}/\Omega \to \Omega$ where Ω is a pointwise ordered object, it suffices to provide a monotone weakest precondition transformer $\mathrm{op}^\Omega : [\![\mathscr{A}]\!] \times \Omega^{[\![\mathscr{B}]\!]} \to \Omega$ that specifies the intended behavior of each $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$.

Example 3.2.3 (Continuation of Example 3.1.16)**.** Let T be the state monad in Example 3.1.9 and $\Omega \coloneqq 2^{\mathbb{Z}}$ be the pointwise ordered object where the order in Ω is defined pointwise: for each $f, g \in 2^{\mathbb{Z}}$, $f \leq g \iff \forall x \in \mathbb{Z}, f(x) \leq g(x)$. We consider the following interpretations of $\mathrm{get} : \mathbf{1} \to \mathbf{int}$ and $\mathrm{set} : \mathbf{int} \to \mathbf{1}$.
$$\mathrm{get}^\Omega(x, p) = \lambda s.p\ s\ s \qquad \mathrm{set}^\Omega(x, p) = \lambda s.p\ ()\ x$$
These interpretations satisfy the effect theory in Example 3.1.9, and it is obvious that these are monotonic in p. Thus, we get a Cartesian lifting of T induced by the above interpretations. This gives the standard weakest precondition transformers for stateful computations.

3.3 Operation-wise Condition for Lifting Algebras

Recall that the interpretation of an effect handler H is the simply fibred T-algebra induced by interpretations of operations. To verify the effect handler H, we need to obtain a lifting of the simply fibred T-algebra. As a program logic for effect handlers, it is desirable to provide a (sufficient) condition of the existence of the lifting as a condition on each operation. We aim to obtain such an operation-wise condition.

In this section, we consider a simpler situation as a warm-up: we consider a lifting of T-algebra (instead of a simply fibred T-algebra) and provide an operation-wise condition. We consider liftings of simply fibred T-algebras later in §3.5 after discussing the problem of strong monad liftings in §3.4.

Definition 3.3.1 (lifting of T-algebra)**.** Let $\alpha : TX \to X$ be a T-algebra in **Set** and $\dot{T}$ be a monad lifting of T along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$. A *lifting* of α is a $\dot{T}$-algebra $\dot{\alpha} : \dot{T}P \to P$ such that $\mathrm{dom}\,\dot{\alpha} = \alpha$.

Since $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$ is faithful, the unit and associative law for $\dot{\alpha}$ follows from those for α.

Lemma 3.3.2. A morphism $\dot{\alpha} : \dot{T}P \to P$ in $\mathbf{Set}/\Omega$ is a lifting of the T-algebra $\alpha : TX \to X$ if and only if $\dot{\alpha}$ is a morphism over α. □

So, we only care about the existence of a morphism over α.

Proposition 3.3.3 (operation-wise condition for lifting T-algebra)**.** Suppose that we have a Cartesian lifting $\dot{T}$ induced by a monotone T-algebra $o := K^T(\Omega, \{\mathrm{op}^\Omega\}_{\mathrm{op}\in\Sigma})$ on the pointwise ordered object Ω. Let $(X, \{\mathrm{op}^X\}_{\mathrm{op}\in\Sigma}) \in \mathbf{Mod}_{\mathcal{T}}$ be a model and $\alpha := K^T(X, \{\mathrm{op}^X\}_{\mathrm{op}\in\Sigma}) \in \mathbf{Set}^T$. We have a $\dot{T}$-algebra on $P \in \mathbf{Set}(X, \Omega)$ over α if and only if

$$\begin{tikzcd}
\llbracket \mathscr{A} \rrbracket \times X^{\llbracket \mathscr{B} \rrbracket} \arrow[r, "\mathrm{op}^X"] \arrow[d, "\llbracket \mathscr{A} \rrbracket \times P^{\llbracket \mathscr{B} \rrbracket}"'] \arrow[dr, phantom, "\leq"] & X \arrow[d, "P"] \\
\llbracket \mathscr{A} \rrbracket \times \Omega^{\llbracket \mathscr{B} \rrbracket} \arrow[r, "\mathrm{op}^\Omega"'] & \Omega
\end{tikzcd} \tag{3.2}$$

holds for each $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$ in Σ.

Proof. We have a $\dot{T}$-algebra on p over α if and only if we have the following inequation by definition.

$$\begin{tikzcd}
TX \arrow[r, "\alpha"] \arrow[d, "TP"'] \arrow[dr, phantom, "\leq"] & X \arrow[d, "P"] \\
T\Omega \arrow[r, "o"'] & \Omega
\end{tikzcd} \tag{3.3}$$

((3.3) $\implies$ (3.2)): For each $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$, we consider a function $f_{\mathrm{op}} : \llbracket \mathscr{A} \rrbracket \times X^{\llbracket \mathscr{B} \rrbracket} \to TX$ defined by $f_{\mathrm{op}}(a, t) := \mathrm{op}_a(\eta \circ t)$. Then we get (3.2) by composing f_{op} and (3.3). Here, we have $\alpha \circ f_{\mathrm{op}} = \mathrm{op}^X$ and $o \circ TP \circ f_{\mathrm{op}} =$

$\mathrm{op}^\Omega \circ (\llbracket \mathscr{A} \rrbracket \times P^{\llbracket \mathscr{B} \rrbracket})$ by the following calculation.

$$\begin{aligned}
\alpha(\mathrm{op}_a(\eta \circ t)) &= \mathrm{op}^X(a, \alpha \circ \eta \circ t) \\
&= \mathrm{op}^X(a, t) \\
o(TP(\mathrm{op}_a(\eta \circ t))) &= o(\mathrm{op}_a(TP \circ \eta \circ t)) \\
&= o(\mathrm{op}_a(\eta \circ P \circ t)) \\
&= \mathrm{op}^\Omega(a, o \circ \eta \circ P \circ t) \\
&= \mathrm{op}^\Omega(a, P \circ t)
\end{aligned}$$

((3.3) $\Longleftarrow$ (3.2)): We prove $o \circ TP \circ x \le P \circ \alpha \circ x$ for each $x : 1 \to TX$ by induction on (a representative of) x.

For the base case, let $x = \eta \circ x'$ for some $x' : 1 \to X$. Then, we have $o \circ TP \circ \eta = P \circ \alpha \circ \eta$ by the following diagram.

$$\begin{tikzcd}
X \arrow[r, "\eta"] \arrow[d, "P"] \arrow[rr, bend left, equal] & TX \arrow[r, "\alpha"] \arrow[d, "TP"] & X \arrow[d, "P"] \\
\Omega \arrow[r, "\eta"] \arrow[rr, bend right, equal] & T\Omega \arrow[r, "o"] & \Omega
\end{tikzcd}$$

Step case: if $x = \mathrm{op}_a(t)$ for some $\mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}$, $a \in \llbracket \mathscr{A} \rrbracket$, and $t \in (TX)^{\llbracket \mathscr{B} \rrbracket}$, then

$$\begin{aligned}
o(TP(\mathrm{op}_a(t))) &= \mathrm{op}^\Omega(a, o \circ TP \circ t) \\
&\le \mathrm{op}^\Omega(a, P \circ \alpha \circ t) \\
&\le P(\mathrm{op}^X(a, \alpha \circ t)) \\
&= P(\alpha(\mathrm{op}_a(t))).
\end{aligned}$$

Here, we use monotonicity of op^Ω and the induction hypothesis in the second line, and we use (3.2) in the third line. □

Note that (3.2) is equivalent to the existence of the following morphism over op^X.

$$\begin{tikzcd}
\mathbf{Set}/\Omega \arrow[d, "\mathrm{dom}"] & \lambda(a,k).\mathrm{op}^\Omega(a, P \circ k) \arrow[r] & P \\
\mathbf{Set} & \llbracket \mathscr{A} \rrbracket \times X^{\llbracket \mathscr{B} \rrbracket} \arrow[r, "\mathrm{op}^X"] & X
\end{tikzcd}$$

3.4 Strong Monad Lifting

The interpretation of our target language uses the strength $\mathrm{st}_{X,Y} : X \times TY \to T(X \times Y)$ of a strong monad T. When we consider a lifting of the interpretation along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$, we need not only a lifting $\dot{T}$ of the monad structure of T but also a lifting of the strength. In this section, we provide a few criteria of the existence of a lifting of the strength and show that some liftings of monads are unfortunately not strong (i.e., have only a "partial" lifting of the strength).

Definition 3.4.1 (strong monad lifting)**.** Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration such that $\mathbb{E}$ and $\mathbb{B}$ have finite products and p strictly preserves finite products. Let T be a strong monad on $\mathbb{B}$. A monad lifting $\dot{T}$ of T is a *strong monad lifting* if $\dot{T}$ is a strong monad and p maps the strength $\dot{\mathrm{st}}$ of $\dot{T}$ to the strength st of T, i.e., $p\ \dot{\mathrm{st}}_{X,Y} = \mathrm{st}_{pX,pY}$ for each $X, Y \in \mathbb{E}$.

Note that $\dot{T}$ being a strong monad lifting is a stronger condition than what we really need for our program logic: to obtain a lifting of the interpretation of a term M, we need only $\dot{\mathrm{st}}_{X,Y}$ for some X, Y that are relevant to the interpretation and pre-/postconditions of M. The existence of a strong monad lifting makes the situation much simpler, but we also consider a program logic that is applicable to the case where there is only a "partial" strength of $\dot{T}$ later in this chapter.

Next, we consider strong monad liftings along the domain fibration $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$. The existence of a strong monad lifting can be reduced to the existence of a specific morphism over the specific component $\mathrm{st}_{\Omega,\Omega} : \Omega \times T\Omega \to T(\Omega \times \Omega)$.

Proposition 3.4.2 (existence of a strong monad lifting along $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$)**.** Assume that $\mathbb{C}$ is a category with finite products and T is a strong monad on $\mathbb{C}$. Assume also that $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$ has fibred finite products. Note that, by [49, Cor. 3.3.6], $\mathbb{C}/\Omega$ has finite products, which are defined by $P \dot{\times} Q := \pi_1^* P \wedge \pi_2^* Q$ where π_1 and π_2 are the first and the second projection, respectively, and $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$ strictly preserves these finite products.

The Cartesian lifting $\dot{T}$ of T induced by a monotone T-algebra $o : T\Omega \to \Omega$ is strong if and only if there is a morphism $\dot{\mathrm{st}}_{\mathrm{id}_\Omega,\mathrm{id}_\Omega} : \mathrm{id}_\Omega \dot{\times} \dot{T}\mathrm{id}_\Omega \to \dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega)$ over $\mathrm{st}_{\Omega,\Omega} : \Omega \times T\Omega \to T(\Omega \times \Omega)$.

$$\begin{tikzcd}
\mathbb{C}/\Omega \arrow[d, "\mathrm{dom}"] & \mathrm{id}_\Omega \dot{\times} \dot{T}\mathrm{id}_\Omega \arrow[r, "\dot{\mathrm{st}}_{\mathrm{id}_\Omega,\mathrm{id}_\Omega}"] & \dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega) \\
\mathbb{C} & \Omega \times T\Omega \arrow[r, "\mathrm{st}_{\Omega,\Omega}"] & T(\Omega \times \Omega)
\end{tikzcd} \tag{3.4}$$

Proof. The essence of the proof is to use the fact that Ω is a split generic object [53, Definition 5.2.1] of $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$.

If $\dot{T}$ is strong, then we obviously have $\dot{\mathrm{st}}_{\mathrm{id}_\Omega,\mathrm{id}_\Omega} : \mathrm{id}_\Omega \dot{\times} \dot{T}\mathrm{id}_\Omega \to \dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega)$ (3.4).

Conversely, assume we have the lifting of (3.4). We first prove that $\mathrm{st}_{X,Y} : X \times TY \to T(X \times Y)$ has a lifting $\dot{\mathrm{st}}_{P,Q} : P \dot{\times} \dot{T}Q \to \dot{T}(P \dot{\times} Q)$ for each $P, Q \in \mathbb{C}/\Omega$. This is given by considering the following diagram.

$$\begin{tikzcd}
 & P \dot{\times} \dot{T}Q \arrow[r, dashed, "\dot{\mathrm{st}}_{P,Q}"] \arrow[d, "\overline{P \times TQ}(\ldots)"'] & \dot{T}(P \dot{\times} Q) \arrow[d, "\overline{T(P \times Q)}(\ldots)"] \\
\mathbb{C}/\Omega \arrow[dd, "\mathrm{dom}"'] & \mathrm{id}_\Omega \dot{\times} \dot{T}\mathrm{id}_\Omega \arrow[r] & \dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega) \\
 & X \times TY \arrow[r, "\mathrm{st}_{X,Y}"] \arrow[d, "P \times TQ"'] & T(X \times Y) \arrow[d, "T(P \times Q)"] \\
\mathbb{C} & \Omega \times T\Omega \arrow[r, "\mathrm{st}_{\Omega,\Omega}"] & T(\Omega \times \Omega)
\end{tikzcd}$$

Here, we use the following equations.

$$(P \times TQ)^*(\mathrm{id}_\Omega \dot{\times} \dot{T}\mathrm{id}_\Omega) = P^*\mathrm{id}_\Omega \dot{\times} (TQ)^*\dot{T}\mathrm{id}_\Omega = P \dot{\times} \dot{T}Q$$

$$(T(P \times Q))^*\dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega) = \dot{T}(P \times Q)^*(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega) = \dot{T}(P^*\mathrm{id}_\Omega \dot{\times} Q^*\mathrm{id}_\Omega) = \dot{T}(P \dot{\times} Q)$$

Lastly, $\dot{\mathrm{st}}$ is natural and satisfies the equations for strengths. This follows because $\mathrm{dom} : \mathbb{C}/\Omega \to \mathbb{C}$ is faithful and $\mathrm{dom}(\dot{\mathrm{st}}) = \mathrm{st}$ is a strength for T. □

The condition (3.4) is equivalent to $\mathrm{id}_\Omega \dot{\times} o \leq o \circ T(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega) \circ \mathrm{st}_{\Omega,\Omega}$ by definition. If Ω is pointwise, then (3.4) is also equivalent to the pointwise inequality

$$(\mathrm{id}_\Omega \dot{\times} o) \circ \langle x, y\rangle \leq o \circ T(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega) \circ \mathrm{st}_{\Omega,\Omega} \circ \langle x, y\rangle$$

for each $x : 1 \to \Omega$ and $y : 1 \to T\Omega$. By [84, Prop. 3.4], this simplifies to the following condition.

Corollary 3.4.3. Consider the situation in Proposition 3.4.2, and assume that Ω is pointwise. The Cartesian lifting $\dot{T}$ is a strong if and only if

$$x \wedge (o \circ y) \leq o \circ T((x \circ !) \wedge \mathrm{id}_\Omega) \circ y. \tag{3.5}$$

holds for each $x : 1 \to \Omega$ and $y : 1 \to T\Omega$. □

There are a few situations that we can easily show the inequality (3.5). Specifically, (3.5) holds if Ω is two-valued.

Lemma 3.4.4. Assume that Ω is pointwise.

- If $\mathbb{C}(1, \Omega)$ has a bottom element $\bot$, then (3.5) holds when $x = \bot$.
- If $\mathbb{C}(1, \Omega)$ has a top element $\top$, then (3.5) holds when $x = \top$.
- If $\mathbb{C}(1, \Omega)$ is isomorphic to $\{\bot \leq \top\}$ as a poset, then $\dot{T}$ is a strong monad lifting.

Proof.
- If $x = \bot \in \mathbb{C}(1, \Omega)$, then (3.5) is $\bot \leq o \circ T(\bot \circ !) \circ y$.
- If $x = \top \in \mathbb{C}(1, \Omega)$, then (3.5) is $o \circ y \leq o \circ y$.
- Apply Corollary 3.4.3. By cases on $x \in \mathbb{C}(1, \Omega) \cong \{\bot \leq \top\}$.

□

Even if Ω itself is not two-valued and $\dot{T}$ is not a strong monad lifting, we may have a lifting of $\mathrm{st}_{X,Y} : X \times TY \to T(X \times Y)$ by taking a predicate $P : X \to \Omega$ from the class of two-valued predicates.

Definition 3.4.5 (two-valued predicate)**.** Assume that Ω is pointwise and that we have a morphism $i : \Omega_2 \to \Omega$ such that (1) $\mathbb{C}(1, \Omega_2) \cong \{\bot, \top\}$ and (2) $\mathbb{C}(1, i) : \mathbb{C}(1, \Omega_2) \to \mathbb{C}(1, \Omega)$ maps $\bot : 1 \to \Omega_2$ to the bottom element $\bot : 1 \to \Omega$ and $\top : 1 \to \Omega_2$ to the top element $\top : 1 \to \Omega$. A predicate $P : X \to \Omega$ is *two-valued* if $P : X \to \Omega$ factors through i, i.e., $P = i \circ P'$ for some $P' : X \to \Omega_2$.

Corollary 3.4.6. If $P : X \to \Omega$ is two-valued, then there is a morphism $\dot{\mathrm{st}}_{P,Q} : P \dot{\times} \dot{T}Q \to \dot{T}(P \dot{\times} Q)$ over $\mathrm{st}_{X,Y} : X \times TY \to T(X \times Y)$.

Proof. Note that we have a morphism $i \dot{\times} o \to \dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega)$ over $\mathrm{st} \circ (i \times \mathrm{id}_{T\Omega})$ by Lemma 3.4.4. The morphism $\dot{\mathrm{st}}_{P,Q}$ is defined by the following diagram.

$$\begin{tikzcd}
P \dot{\times} \dot{T}Q \arrow[r, dashed, "\dot{\mathrm{st}}_{P,Q}"] \arrow[d, "\overline{P' \times TQ}(\dots)"'] & \dot{T}(P \dot{\times} Q) \arrow[d, "\overline{T(P \times Q)}(\dots)"] \\
i \dot{\times} o \arrow[r] & \dot{T}(\mathrm{id}_\Omega \dot{\times} \mathrm{id}_\Omega)
\end{tikzcd}$$

$$\begin{tikzcd}
X \times TY \arrow[r, "\mathrm{st}_{X,Y}"] \arrow[d, "P' \times TQ"'] & T(X \times Y) \arrow[d, "T(P \times Q)"] \\
\Omega' \times T\Omega \arrow[r, "\mathrm{st}_{\Omega,\Omega} \circ (i \times \mathrm{id}_{T\Omega})"] & T(\Omega \times \Omega)
\end{tikzcd}$$

□

Example 3.4.7 ($\Omega = 2^{\mathbb{Z}}$)**.** If T is the state monad in Example 3.1.9 and $\Omega = 2^{\mathbb{Z}}$ (see Example 3.2.3), then $i : \Omega_2 \to \Omega$ is given by the inclusion $\Omega_2 \coloneqq \{\lambda s.\bot, \lambda s.\top\} \subseteq \Omega$. That is, $P : X \to \Omega$ is two-valued if and only if $P(x)(s) \in 2$ does not depend on $s \in \mathbb{Z}$.

We can decompose the condition in Corollary 3.4.3 to obtain an operation-wise condition.

Lemma 3.4.8 (operation-wise condition for lifting a strength)**.** Consider the situation in Proposition 3.4.2. Let $\mathbb{C}$ be **Set** and T be a monad induced by an effect theory $\mathcal{T}$. Assume that Ω is a pointwise ordered object and that $(\Omega, \{\mathrm{op}^\Omega\}) \in \mathbf{Mod}_{\mathcal{T}}$ is monotone in the sense of Theorem 3.2.1. The Cartesian lifting $\dot{T}$ induced by $o \coloneqq K^T(\Omega, \{\mathrm{op}^\Omega\})$ is a strong monad lifting of T if and only if

$$x \wedge \mathrm{op}^\Omega(a, y) \le \mathrm{op}^\Omega(a, (x \circ !) \wedge y) \tag{3.6}$$

holds for each $\mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}$, $x : 1 \to \Omega$, $y : [\![\mathscr{B}]\!] \to \Omega$, and $a \in [\![\mathscr{A}]\!]$.

Proof. If we have a strong monad lifting, then we obtain (3.6) by substituting $\mathrm{op}_a(\eta \circ y')$ for y in (3.5) where $y' : [\![\mathscr{B}]\!] \to \Omega$.

For the converse, we prove (3.5) by induction on (a representative of) $y : 1 \to TY$. For the base case where $y = \eta \circ y'$ for some $y' : 1 \to \Omega$, (3.5) is $x \wedge y' \le x \wedge y'$, which trivially holds. For the step case, let $y = \mathrm{op}_a(t)$ for some $a \in [\![\mathscr{A}]\!]$ and $t \in (T\Omega)^{[\![\mathscr{B}]\!]}$.

$$\begin{aligned}
x \wedge (o \circ \mathrm{op}_a(t)) &= x \wedge \mathrm{op}^\Omega(a, o \circ t) \\
&\le \mathrm{op}^\Omega(a, (x \circ !) \wedge (o \circ t)) && \text{by (3.6)} \\
&\le \mathrm{op}^\Omega(a, o \circ T((x \circ !) \wedge \mathrm{id}_\Omega) \circ t) && \text{by IH (see below)} \\
&= o \circ T((x \circ !) \wedge \mathrm{id}_\Omega) \circ \mathrm{op}^\Omega(a, t)
\end{aligned}$$

Note that for each $b : 1 \to [\![\mathscr{B}]\!]$, we have by the induction hypothesis

$$((x \circ !) \wedge (o \circ t)) \circ b = x \wedge (o \circ t \circ b) \le o \circ T((x \circ !) \wedge \mathrm{id}_\Omega) \circ t \circ b,$$

and thus, we have $(x \circ !) \wedge (o \circ t) \le o \circ T((x \circ !) \wedge \mathrm{id}_\Omega) \circ t$. □

The inequation (3.6) can be understood as something similar to *the rule of constancy* in Hoare logic. Since op^Ω is the weakest precondition transformer of the generic effect $\mathrm{gen}_{\mathrm{op}}$ by Proposition 3.2.2, the inequation (3.6) can be read as $(P \circ !) \wedge \mathrm{wp}[\mathrm{gen}_{\mathrm{op}}](Q) \le \mathrm{wp}[\mathrm{gen}_{\mathrm{op}}]((P \circ !) \wedge Q)$ for each $\mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}$,

$P : 1 \to \Omega$, and $Q : [\![\mathscr{B}]\!] \to \Omega$. By Definition 3.1.6, it is equivalent to the following rule for Hoare triples.

$$\textsc{constancy}\ \frac{\{R\}\mathrm{gen}_{\mathrm{op}}\{Q\}}{\{(P \circ !) \wedge R\}\mathrm{gen}_{\mathrm{op}}\{(P \circ !) \wedge Q\}}$$

Here, $P \circ !$ represents a predicate that does not refer to neither $[\![\mathscr{A}]\!]$ nor $[\![\mathscr{B}]\!]$ in this rule.

Example 3.4.9. Consider $\mathrm{dom} : \mathbf{Set}/\mathbf{2} \to \mathbf{Set}$ where $\mathbf{2} = \{\bot \leq \top\}$ is the pointwise ordered object. Any Cartesian lifting of a monad T on **Set** along $\mathrm{dom} : \mathbf{Set}/\mathbf{2} \to \mathbf{Set}$ is a strong monad lifting by Lemma 3.4.4. This situation includes

- two Cartesian liftings of the maybe monad that represent *total* and *partial* correctness [8, Example 5.1],
- two Cartesian liftings of the nonempty finite powerset monad that represent *may* and *must* correctness of nondeterministic computation [8, Example 5.2], and
- two Cartesian liftings of the finite probability distribution monad that represent *may* and *must* correctness of probabilistic computation [8, Theorem 5.7].

Example 3.4.10. Let $T \coloneqq D$ be the finitely supported probability distribution monad and $\Omega \coloneqq [0, \infty]$. We have a monotone T-algebra $\mathbb{E} : T\Omega \to \Omega$ defined by the expectation operator [8, Example 6.3]. However, $\mathbb{E}$ does not satisfy (3.5).

The inequality (3.5) is equivalent to $x \wedge \mathbb{E}\mu \leq \mathbb{E}(\mu \downarrow x)$ where $x \in [0, \infty]$, $\mu = y \in D[0, \infty]$, and $\mu \downarrow x$ is the "truncated" probability distribution defined by

$$(\mu \downarrow x)(z) = \begin{cases} \mu(z) & \text{if } z < x \\ \Sigma_{z' \geq x}\mu(z') & \text{if } z = x \\ 0 & \text{if } z > x. \end{cases}$$

A counterexample is given by $x = 1$ and μ defined by $\mu(0) = \mu(2) = \frac{1}{2}$. Here, the truncated probability distribution $\mu \downarrow x$ is given by $(\mu \downarrow x)(0) = (\mu \downarrow x)(1) = \frac{1}{2}$. Therefore, we have $x \wedge \mathbb{E}\mu = 1 \not\leq \mathbb{E}(\mu \downarrow x) = \frac{1}{2}$.

Example 3.4.11. Let $T \coloneqq (S \times (-))^S$ be the state monad and $\Omega \coloneqq 2^S$ where $S \in \mathbf{Set}$. We have a monotone EM T-algebra $o : T\Omega \to \Omega$ defined by $o(f) \coloneqq \lambda s.\mathbf{let}\ (s', p) = f\ s\ \mathbf{in}\ p\ s'$. However, o does not satisfy (3.5) if $|S| \geq 2$ by the following argument.

Let $s_1, s_2 \in S$ be states such that $s_1 \neq s_2$. Let $x = (\lambda s.s = s_1) \in 2^S$ and $y = \lambda s.(s_2, \lambda s'.s' = s_2) \in (S \times 2^S)^S$. Then, the lhs of (3.5) is $\lambda s.s = s_1$ while the rhs is $\lambda s.\bot$, and we have $\lambda s.(s = s_1) \not\leq \lambda s.\bot$.

3.5 Operation-wise Condition for Lifting Simply Fibred Algebras

In this section, we consider a lifting of a simply fibred T-algebra. If we have a strong monad lifting, a sufficient condition for the existence of the lifting of a simply fibred T-algebra is a straightforward extension of Proposition 3.3.3 If we do not have a strong monad lifting, that is, if a lifting $\dot{\mathrm{st}}_{P,Q} : P \dot{\times} \dot{T}Q \to \dot{T}(P \dot{\times} Q)$ of $\mathrm{st}_{X,Y} : X \times TY \to T(X \times Y)$ does not necessarily exist, then we need to restrict the predicate P to the class of two-valued predicates to guarantee the existence of $\dot{\mathrm{st}}_{P,Q}$.

Definition 3.5.1 (lifting of simply fibred T-algebra)**.** Let $\alpha : I \times TX \to X$ be a simply fibred T-algebra and $\dot{T}$ be a monad lifting along $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$. A *lifting* of α is a simply fibred $\dot{T}$-algebra $\dot{\alpha} : P \dot{\times} \dot{T}Q \to Q$ such that $\mathrm{dom}\, \dot{\alpha} = \alpha$. Here, even if the strength $\dot{\mathrm{st}}$ of $\dot{T}$ is only partially defined, we say $\dot{\alpha}$ is a simply fibred $\dot{T}$-algebra if $\dot{\mathrm{st}}_{P,\dot{T}Q} : P \dot{\times} \dot{T}^2 Q \to \dot{T}(P \dot{\times} \dot{T}Q)$ is defined and $\dot{\alpha}$ satisfies the condition in Definition 3.1.13.

Assuming that $\dot{\mathrm{st}}_{P,\dot{T}Q} : P \dot{\times} \dot{T}^2 Q \to \dot{T}(P \dot{\times} \dot{T}Q)$ is defined, $\dot{\alpha}$ is a lifting of α if and only if $\dot{\alpha}$ is a morphism over α by a similar argument to (not simply fibred) T-algebras.

Proposition 3.5.2 (operation-wise lifting of fibred T-algebra)**.** Let T be a monad on $\mathbf{Set}$ induced by an effect theory $\mathcal{T}$, $\dot{T}$ be a Cartesian lifting of T induced by a monotone T-algebra $o := K^T(\Omega, \{\mathrm{op}^\Omega\})$ on the pointwise ordered object Ω, and $\alpha := K^T_I(X, \{\mathrm{op}^X\})$ be a simply fibred T-algebra.

1. Consider the case where $\dot{T}$ is strong. The morphism $\alpha : I \times TX \to X$ has a lifting $\dot{\alpha} : P \dot{\times} \dot{T}Q \to Q$ if for all $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$, the following inequation holds
$$\begin{tikzcd} I \times [\![\mathscr{A}]\!] \times X^{[\![\mathscr{B}]\!]} \arrow[r, "\mathrm{op}^X"] \arrow[d, "I \times [\![\mathscr{A}]\!] \times Q^{[\![\mathscr{B}]\!]}"'] \arrow[dr, phantom, "\leq"] & X \arrow[d, "Q"] \\ I \times [\![\mathscr{A}]\!] \times \Omega^{[\![\mathscr{B}]\!]} \arrow[r, "P \dot{\times} \mathrm{op}^\Omega"'] & \Omega \end{tikzcd} \tag{3.7}$$
that is, there exists a morphism $\dot{\mathrm{op}}^X : (P \dot{\times} \mathrm{op}^\Omega) \circ (I \times [\![\mathscr{A}]\!] \times Q^{[\![\mathscr{B}]\!]}) \to Q$ over op^X in $\mathrm{dom} : \mathbf{Set}/\Omega \to \mathbf{Set}$.

2. Consider the case where $\dot{T}$ is not necessarily strong. The morphism $\alpha : I \times TX \to X$ has a lifting $\dot{\alpha} : P \dot{\times} \dot{T}Q \to Q$ if P is two-valued and for all $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$, (3.7) holds.

Proof. 1. If $\dot{T}$ is strong, we prove
$$(P \dot{\times} \dot{T}Q) \circ \langle x, y \rangle \leq Q \circ \alpha \circ \langle x, y \rangle$$
for each $x : 1 \to X$ and $y : 1 \to TY$ by induction on (a representative of) y.

For the base case, let $y = \eta \circ y'$ for some $y' : 1 \to Y$.
$$\begin{aligned} (P \dot{\times} \dot{T}Q) \circ \langle x, y \rangle &= (P \circ x) \wedge (o \circ TQ \circ \eta \circ y') \\ &= (P \circ x) \wedge (Q \circ y') \\ &\leq Q \circ y' \\ &= Q \circ \alpha \circ \langle x, y \rangle \end{aligned}$$

For the step case, let $y = \mathrm{op}_a(t)$ for some $\mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}$, $a \in [\![\mathscr{A}]\!]$, and $t : [\![\mathscr{B}]\!] \to TY$.

$$\begin{aligned}
&(P \dot{\times} \dot{T} Q) \circ \langle x, y \rangle \\
&= (P \circ x) \wedge (o \circ TQ \circ \mathrm{op}_a(t)) \\
&= (P \circ x) \wedge \mathrm{op}^{\Omega}(a, o \circ TQ \circ t) \\
&\leq (P \circ x) \wedge \mathrm{op}^{\Omega}(a, (P \circ x \circ !) \wedge (o \circ TQ \circ t)) && \text{by Lemma 3.4.8} \\
&\leq (P \circ x) \wedge \mathrm{op}^{\Omega}(a, Q \circ \alpha \circ \langle x \circ !, t \rangle) && \text{by IH} \\
&\leq Q \circ \mathrm{op}^{X,Y}(x, a, \alpha \circ \langle x \circ !, t \rangle) && \text{by assumption} \\
&= Q \circ \alpha \circ \langle x, \mathrm{op}_a(t) \rangle \\
&= Q \circ \alpha \circ \langle x, y \rangle
\end{aligned}$$

2. The proof is almost the same as above except that we cannot use Lemma 3.4.8 in this case. Here, we prove

$$(P \circ x) \wedge \mathrm{op}^{\Omega}(a, o \circ TQ \circ t) \leq \mathrm{op}^{\Omega}(a, (P \circ x \circ !) \wedge (o \circ TQ \circ t))$$

by cases. There are two cases: $P \circ x$ is either $\top$ or $\bot$ because P is two-valued. In both cases, the inequality follows immediately.

□

3.6 Program Logic

In this section, we consider a program logic for effect handlers. We provide inference rules for generic effects, effect handlers, and handling constructs and prove the soundness of inference rules. We do not consider inference rules for the other construct of the language (such as lambda abstractions and applications) because these are not our aim and are already studied in e.g. [7].

Judgement. In our program logic, we use three kinds of judgements for value terms, computation terms, and effect handlers.

$$\begin{gathered}
\Gamma \mid K \ \{P\} \vdash V : A \ \{Q\} \\
\Gamma \mid K \ \{P\} \vdash M : \underline{C} \ \{R\} \\
\Gamma \mid K \ \{P\} \vdash H : \underline{C} \ \{R\} \ \textbf{handler}
\end{gathered}$$

Here, typing judgements are annotated with the precondition $P : [\![\Gamma]\!] \times [\![K]\!] \to \Omega$ and the postconditions $Q : [\![A]\!] \to \Omega$ and $R : U[\![\underline{C}]\!] \to \Omega$ where Ω is a pointwise ordered object in **Set**.

The meaning of these judgements are as follows. Let T be a monad on **Set** induced by an effect theory $\mathcal{T}$ and $\dot{T}$ be a Cartesian lifting of T induced by $o \coloneqq K^T(\Omega, \{\mathrm{op}^{\Omega}\})$. For value terms, $\Gamma \mid K \ \{P\} \vdash V : A \ \{Q\}$ means that there is a lifting $P \to Q$ of $[\![V]\!]$.

$$\begin{tikzcd}
\mathbf{Set}/\Omega \arrow[d, "\mathrm{dom}"] & P \arrow[r] & Q \\
\mathbf{Set} & {[\![\Gamma]\!] \times [\![K]\!]} \arrow[r, "{[\![V]\!]}"] & {[\![A]\!]}
\end{tikzcd}$$

$$\dfrac{\textsc{GenericEffect} \qquad \mathrm{op} : \mathscr{A} \to \mathscr{B} \qquad \Gamma \mid K\ \{P\} \vdash V : \mathscr{A}\ \{\mathrm{op}^{\Omega}(-, Q)\}}{\Gamma \mid K\ \{P\} \vdash \mathrm{gen}_{\mathrm{op}}(V) : F\mathscr{B}\ \{\dot{T}Q\}}$$

where $\mathrm{gen}_{\mathrm{op}}(V) \coloneqq \mathrm{op}_V(o.\mathbf{return}\ o)$.

Figure 3.2: The inference rule for generic effects.

The meaning for computation terms $\Gamma \mid K\ \{P\} \vdash M : \underline{C}\ \{R\}$ is defined similarly. For effect handlers, $\Gamma \mid K\ \{P\} \vdash H : \underline{C}\ \{R\}$ **handler** means that there is a lifting of $[\![H]\!]$ of the following form.

$$\begin{tikzcd} \mathbf{Set}/\Omega \arrow[d, "\mathrm{dom}"] & P\dot{\times}\dot{T}R \arrow[r] & R \\ \mathbf{Set} & ([\![\Gamma]\!] \times [\![K]\!]) \times TU[\![\underline{C}]\!] \arrow[r, "{[\![H]\!]}"] & U[\![\underline{C}]\!] \end{tikzcd}$$

3.6.1 Generic Effects

By the equivalence between generic effects and algebraic operations [91], we can rewrite algebraic operations to generic effects. So, we only consider generic effects for simplicity. The inference rule is shown in Fig. 3.2.

Theorem 3.6.1 (soundness)**.** GenericEffect is sound.

Proof. The interpretation of $\mathrm{gen}_{\mathrm{op}}(V)$ is given by $[\![\mathrm{gen}_{\mathrm{op}}(V)]\!] = \mathrm{gen}_{\mathrm{op}} \circ [\![V]\!]$ where the $\mathrm{gen}_{\mathrm{op}} : [\![\mathscr{A}]\!] \to T[\![\mathscr{B}]\!]$ is defined in Proposition 3.2.2. Therefore, we have the following lifting.

$$\begin{tikzcd} \mathbf{Set}/\Omega \arrow[d, "\mathrm{dom}"] & P \arrow[r] & \mathrm{op}^{\Omega}(-, Q) \arrow[r] & \dot{T}Q \\ \mathbf{Set} & [\![\Gamma]\!] \times [\![K]\!] \arrow[r, "{[\![V]\!]}"] & [\![\mathscr{A}]\!] \arrow[r, "\mathrm{gen}_{\mathrm{op}}"] & [\![\mathscr{B}]\!] \end{tikzcd}$$

Here, the left lifting is the premise of the rule, and the right lifting is by Proposition 3.2.2. □

Note that the soundness of the rule for generic effects does not depend on whether there exists a strong monad lifting of T.

3.6.2 With a Strong Monad Lifting

First, we consider the case where the Cartesian lifting $\dot{T}$ is strong. We can verify effect handlers operation-wise. Inference rules of our program logic are given in Figure 3.3.

Theorem 3.6.2 (soundness)**.** If $\dot{T}$ is strong, then Handler-Strong and Handle-Strong are sound.

Proof. For Handler-Strong, apply Proposition 3.5.2 to $[\![H]\!]$ in Definition 3.1.15.

Because every ingredient of $[\![M\ \textbf{handled with}\ H\ \textbf{to}\ x{:}A\ \textbf{in}\ N]\!]$ in Definition 3.1.15 has a lifting, Handle-Strong follows. □

Handler-Strong

$$\frac{\begin{array}{l}\forall \mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}, \\ \Gamma, x : \mathscr{A} \mid K, k : \mathscr{B} \to \underline{C}\ \{\lambda((\gamma, x), (\kappa, k)).P(\gamma, \kappa) \wedge \mathrm{op}^{\Omega}(x, Q \circ k)\} \vdash M_{\mathrm{op}} : \underline{C}\ \{Q\}\end{array}}{\Gamma \mid K\ \{P\} \vdash \{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\} : \underline{C}\ \{Q\}\ \textbf{handler}}$$

Handle-Strong

$$\frac{\begin{array}{c}\Gamma \mid K\ \{P\} \vdash M : FA\ \{\dot{T}Q\} \qquad \Gamma \mid K\ \{P\} \vdash H : \underline{C}\ \{R\}\ \textbf{handler} \\ \Gamma, x : A \mid K\ \{\lambda((\gamma, x), \kappa).P(\gamma, \kappa) \wedge Q(x)\} \vdash N : \underline{C}\ \{R\}\end{array}}{\Gamma \mid K\ \{P\} \vdash M\ \textbf{handled with}\ H\ \textbf{to}\ x{:}A\ \textbf{in}\ N : \underline{C}\ \{R\}}$$

Figure 3.3: Inference rules when we have a strong monad lifting.

Handler-Weak

$$\frac{\begin{array}{c}\forall \mathrm{op} : \mathscr{A} \rightarrowtriangle \mathscr{B}, \\ \Gamma, x : \mathscr{A} \mid K, k : \mathscr{B} \to \underline{C}\ \{\lambda((\gamma, x), (\kappa, k)).P(\gamma, \kappa) \wedge \mathrm{op}^{\Omega}(x, Q \circ k)\} \vdash M_{\mathrm{op}} : \underline{C}\ \{Q\} \\ P \text{ is two-valued}\end{array}}{\Gamma \mid K\ \{P\} \vdash \{\mathrm{op}(x; k) \mapsto M_{\mathrm{op}}\} : \underline{C}\ \{Q\}\ \textbf{handler}}$$

Handle-Weak

$$\frac{\begin{array}{c}\Gamma \mid K\ \{P\} \vdash M : FA\ \{\dot{T}Q\} \qquad \Gamma \mid K\ \{P'\} \vdash H : \underline{C}\ \{R\}\ \textbf{handler} \\ \Gamma, x : A \mid K\ \{\lambda((\gamma, x), \kappa).P'(\gamma, \kappa) \wedge Q(x)\} \vdash N : \underline{C}\ \{R\} \\ P \leq P' \qquad P' \text{ is two-valued}\end{array}}{\Gamma \mid K\ \{P\} \vdash M\ \textbf{handled with}\ H\ \textbf{to}\ x{:}A\ \textbf{in}\ N : \underline{C}\ \{R\}}$$

Figure 3.4: Inference rules when we do not have a strong monad lifting.

3.6.3 Without a Strong Monad Lifting

Next, we consider the case where the Cartesian lifting $\dot{T}$ is not necessarily strong. We cannot use inference rules in Figure 3.3, but if we consider two-valued predicates, then we obtain similar rules shown in Figure 3.4.

Theorem 3.6.3 (soundness)**.** Assume we have $i : \Omega_2 \to \Omega$ in Definition 3.4.5. Handler-Weak and Handle-Weak are sound.

Proof. For Handler-Weak, apply Proposition 3.5.2 to $[\![H]\!]$ in Definition 3.1.15.

Because every ingredient of $[\![M\ \textbf{handled with}\ H\ \textbf{to}\ x{:}A\ \textbf{in}\ N]\!]$ in Definition 3.1.15 has a lifting, Handle-Weak follows. Here, we use Corollary 3.4.6 for the existence of a lifting of $\mathrm{st}_{[\![\Gamma]\!] \times [\![K]\!], [\![A]\!]}$. □

Example 3.6.4 (Continuation of Example 3.2.3)**.** Let H be the effect handler in Example 3.1.11 and consider proving

$$\diamond \mid \diamond\ \{\top\} \vdash H : (\textbf{int} \to F\ \textbf{int})\ \{Q\}\ \textbf{handler}$$

where $Q : U[\![\mathbf{int} \to F\ \mathbf{int}]\!] \to \Omega$. By Handler-Weak, it suffices to prove

$$x : \mathbf{int} \mid k : \mathbf{1} \to (\mathbf{int} \to F\ \mathbf{int})\ \{\mathrm{set}^{\Omega}(x, Q \circ k)\} \vdash \lambda s.k\ ()\ x : \mathbf{int} \to F\ \mathbf{int}\ \{Q\}$$
$$x : \mathbf{1} \mid k : \mathbf{int} \to (\mathbf{int} \to F\ \mathbf{int})\ \{\mathrm{get}^{\Omega}(x, Q \circ k)\} \vdash \lambda s.k\ s\ s : \mathbf{int} \to F\ \mathbf{int}\ \{Q\}$$

which are equivalent to

$$\begin{array}{ll} \forall x \in \mathbb{Z}, \forall k : 1 \to \mathbb{Z} \to UF\mathbb{Z}, & \lambda s.Q\ (k\ ())\ x \le Q\ (\lambda s.k\ ()\ x) \\ \forall k : \mathbb{Z} \to \mathbb{Z} \to UF\mathbb{Z}, & \lambda s.Q\ (k\ s)\ s \le Q\ (\lambda s.k\ s\ s). \end{array}$$

These are satisfied if we define Q by $Q\ f\ s \coloneqq Q'\ (f\ s)$ for some $Q' : UF\mathbb{Z} \to 2$.

Next, we consider

$$\diamond \mid \diamond \vdash M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x{:}\mathbf{int}\ \mathbf{in}\ N : \mathbf{int} \to F\ \mathbf{int}$$

where

$$M \coloneqq \mathrm{get}((); \lambda s.\mathbf{return}\ s) \qquad N \coloneqq \lambda s.\mathbf{return}\ x$$

and prove that this term is a function that returns 0 when the input is 0. Specifically, we prove

$$\diamond \mid \diamond\ \{\lambda s.s = 0\} \vdash M\ \mathbf{handled\ with}\ H\ \mathbf{to}\ x{:}\mathbf{int}\ \mathbf{in}\ N : \mathbf{int} \to F\ \mathbf{int}\ \{\lambda r.\lambda s.r\ s = \eta\ 0\}$$

where η is the unit of T. Note that this judgement is a bit tricky: it says that if the initial state is $s = 0$ and if we apply this term to the initial state s, then the result is $\eta\ 0$ which is a pure computation that returns 0.

By Handle-Weak, it suffices to prove

$$\diamond \mid \diamond\ \{\lambda s.s = 0\} \vdash M : F\ \mathbf{int}\ \{\dot{T}(\lambda x.\lambda s.x = 0)\}$$
$$\diamond \mid \diamond\ \{\top\} \vdash H : (\mathbf{int} \to F\ \mathbf{int})\ \{\lambda r.\lambda s.r\ s = \eta\ 0\}\ \mathbf{handler}$$
$$x : \mathbf{int} \mid \diamond\ \{\lambda x.\lambda s.x = 0\} \vdash N : \mathbf{int} \to F\ \mathbf{int}\ \{\lambda r.\lambda s.r\ s = \eta\ 0\}.$$

Note that the second judgement follows from the above argument. The third judgement is also obvious.

We apply GenericEffect to the first judgement. The proof obligation is now $\diamond \mid \diamond\ \{\lambda s.s = 0\} \vdash () : \mathbf{1}\ \{\lambda s.s = 0)\}$, which is obviously true.

3.7 Related Work

Logics for computational effects. A fibrational account of weakest precondition transformers is presented using monad liftings in [8]. They also discuss the relationship between monad liftings and Eilenberg-Moore algebras.

There is another construction of monad liftings called $\top\top$-lifting [63]. $\top\top$-liftings are applied to the *effect simulation problem* [64] for a language with algebraic operation. Program logics for differential privacy is also studied in [99, 100] using $\top\top$-liftings as semantic foundation.

However, none of the above papers deal with effect handlers.

Verification of effect handlers. There are not many studies on verifying effect handlers. Some of them consider equational reasoning of whether a given effect handler satisfies an effect theory. Plotkin and Pretnar [92, Section 5] focus on *existence assertion* $M\downarrow$ and *Kleene equality* $M \simeq N$ to reason about effect handlers. Equational reasoning is further studied in [80], which provides a more practical type system that can track equations.

More general predicates than just equations are studied in [10, Section 7]. Here, effect handlers are used to lift a predicate $P : A \to \Omega$ on the value type A to a predicate $P' : UFA \to \Omega$ on the type UFA. This is similar to what we did in 3.2 to define a Cartesian lifting, but they did not provide the clear correspondence between a T-algebra on Ω and a weakest precondition transformer of each generic effect (Theorem 3.2.1 and Proposition 3.2.2) nor the program logic (Section 3.6) to verify effect handlers.

A separation logic for effect handlers is proposed in [31]. Their separation logic is based on Iris [58]. They define *protocols* that specify preconditions and postconditions of algebraic operations and then define the weakest precondition transformer of the form $\mathrm{ewp}\ e\ \langle\Psi\rangle\{\Phi\}$ where e is a program, Ψ is a protocol, and Φ is a postcondition. The main difference from our work is that their effect handlers are limited to one-shot continuations with only one unnamed effect. That is, their effect handler cannot invoke a continuation twice nor use multiple algebraic operations (e.g. set and get for stateful computations) at the same time.

The most related paper to ours is [81, Section 6]. They provide two approaches to verify effect handlers in a dependently typed functional language, and specifically their "2nd approach" is similar to our program logic as explained below. Given a specification $(P_{\mathrm{op}}, Q_{\mathrm{op}})$ for each operation $\mathrm{op} : I \rightarrowtriangle O$ where $P_{\mathrm{op}} : I \to \mathbb{P}$ is a precondition, $Q_{\mathrm{op}} : I \times O \to \mathbb{P}$ is a postcondition, and $\mathbb{P}$ is a type of predicates, then their 2nd approach verifies effect handlers by considering the following handling construct.

$$\begin{aligned}
\mathrm{handle} : {} & \mathcal{D}\ A\ (\lambda p.\forall a.Q\ a \to p\ a) \\
& \to ((a : A) \to Q\ a \to (b : B) \times R\ b) \\
& \to (h_{\mathrm{op}} : I \to (O \to B) \to B)_{\mathrm{op}:I \rightarrowtriangle O} \\
& \to (\forall i.\forall k.(\forall o.Q_{\mathrm{op}}\ (i,o) \to R\ (k\ o)) \to P_{\mathrm{op}}\ i \to R\ (h_{\mathrm{op}}\ i\ k))_{\mathrm{op}:I \rightarrowtriangle O} \\
& \to (b : B) \times R\ b
\end{aligned} \tag{3.8}$$

This can be understood in our approach as follows. For each operation $\mathrm{op} : I \rightarrowtriangle O$, suppose that we are given a specification $(P_{\mathrm{op}}, Q_{\mathrm{op}})$ where $P_{\mathrm{op}} : [\![I]\!] \to \Omega$ and $Q_{\mathrm{op}} : [\![I]\!] \times [\![O]\!] \to \Omega$, and Ω is a pointwise ordered object. Consider a monotone T-algebra $T\Omega \to \Omega$ induced by $\mathrm{op}^{\Omega}(a,k) \coloneqq P_{\mathrm{op}}\ a \wedge \forall b.Q_{\mathrm{op}}\ (a,b) \implies k\ o$. Using the Cartesian lifting induced by this monotone T-algebra, the combination of our Handle-Strong and Handler-Strong rules applied to the judgement $\Gamma \mid K\ \{\top\} \vdash M$ **handled with** $\{\mathrm{op}(x;k) \mapsto h_{\mathrm{op}}\}$ **to** $x{:}A$ **in** $N : \underline{C}\ \{R\}$ roughly corresponds to the above handling construct.

$$\dfrac{J_1 \qquad J_2 \qquad \dfrac{J_3}{\Gamma \mid K\ \{\top\} \vdash \{\mathrm{op}(x;k) \mapsto h_{\mathrm{op}}\} : \underline{C}\ \{R\}\ \textbf{handler}}}{\Gamma \mid K\ \{\top\} \vdash M\ \textbf{handled with}\ \{\mathrm{op}(x;k) \mapsto h_{\mathrm{op}}\}\ \textbf{to}\ x{:}A\ \textbf{in}\ N : \underline{C}\ \{R\}}$$

Here, J_1, J_2, and J_3 are judgements defined as follows.

$$
\begin{aligned}
J_1 &\coloneqq \Gamma \mid K \; \{\top\} \vdash M : FA \; \{\dot{T}Q\} \\
J_2 &\coloneqq \Gamma, x : A \mid K \; \{Q \; x\} \vdash N : \underline{C} \; \{R\} \\
J_3 &\coloneqq \forall \mathrm{op} : \mathscr{A} \twoheadrightarrow \mathscr{B}, \; \Gamma, x : \mathscr{A} \mid K, k : \mathscr{B} \to \underline{C} \; \{\mathrm{op}^{\Omega}(x, R \circ k)\} \vdash h_{\mathrm{op}} : \underline{C} \; \{R\}
\end{aligned}
$$

The first argument of handle (3.8) corresponds to the judgement J_1 for the computation M being handled, the second argument corresponds to the judgement J_2 for the computation N for returned value, the third argument corresponds to the effect handler $H = \{\mathrm{op} \mapsto h_{\mathrm{op}}\}$, the fourth argument corresponds to the judgement J_3, which is the premise of Handler-Strong (recall the definition of op^{Ω} above), and the result type corresponds to the postcondition R of the whole term. They say that their approach is "ad hoc", but we provide a theoretical understanding via liftings along fibrations. Moreover, our fibrational approach provides clues to the generalization to various monads. For example, our approach can deal with the state monad, which is left unsolved in their paper.

Chapter 4

Decision Tree-Based Ranking Function Synthesis

In this chapter, we provide an example-based synthesis of ranking functions for termination analysis.

4.1 Preview by Examples

We present a preview of our method using concrete examples. We start with an overview of the general CEGIS architecture, after which we proceed to our main contribution, namely a decision tree learning algorithm for transition examples.

4.1.1 Termination Verification by CEGIS

Our method follows the usual workflow of termination verification by CEGIS. It works as follows: given a program, we encode the termination problem into a constraint solving problem, and then use the CEGIS architecture to solve the constraint solving problem.

Encoding the termination problem. The first step of our method is to encode the termination problem as the set $\mathcal{C}$ of constraints.

Example 4.1.1. As a running example, consider the following C program.

```
while(x != 0) { if(x < 0) { x++; } else { x--; } }
```

The termination problem is encoded as the following constraints.

$$x < 0 \land x' = x + 1 \implies R(x, x') \tag{4.1}$$

$$\neg(x < 0) \land x' = x - 1 \implies R(x, x'). \tag{4.2}$$

Here, R is a predicate variable representing a well-founded relation, and term variables x, x' are universally quantified implicitly.

The set $\mathcal{C}$ of constraints claims that the transition relation for the given program is subsumed by a well-founded relation. So, verifying termination is now rephrased as the existence of a solution for $\mathcal{C}$. Note that we omitted constraints for invariants for simplicity in this example (see Section 4.2 for the full encoding).

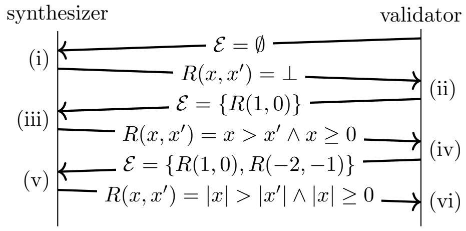

synthesizer
validator
(i)
$\mathcal{E} = \emptyset$
$R(x, x') = \bot$
(ii)
(iii)
$\mathcal{E} = \{R(1, 0)\}$
$R(x, x') = x > x' \land x \geq 0$
(iv)
(v)
$\mathcal{E} = \{R(1, 0), R(-2, -1)\}$
$R(x, x') = |x| > |x'| \land |x| \geq 0$
(vi)

Figure 4.1: An example of CEGIS iterations

Constraint solving by CEGIS. The next step is to solve $\mathcal{C}$ by CEGIS.

In the CEGIS architecture, a synthesizer and a validator iteratively exchange a set $\mathcal{E}$ of examples and a candidate solution $R(x, x')$ for $\mathcal{C}$. At the moment, we present a rough sketch of CEGIS, leaving the details of our implementation to Section 4.1.2.

Example 4.1.2. Fig. 4.1 shows how the CEGIS architecture solves the set $\mathcal{C}$ of constraints shown in (4.1) and (4.2). Fig. 4.1 consists of three pairs of interactions (i)-(vi) between a synthesizer and a validator.

(i) The synthesizer takes $\mathcal{E} = \emptyset$ as a set of examples and returns a candidate solution $R(x, x') = \bot$ synthesized from $\mathcal{E}$. In general, candidate solutions are required to satisfy all constraints in $\mathcal{E}$, but the requirement is vacuously true in this case.

(ii) The validator receives the candidate solution and finds out that the candidate solution is not a genuine solution. The validator finds that the assignment $x = 1, x' = 0$ is a counterexample for (4.2), and thus adds $R(1, 0)$ to $\mathcal{E}$ to prevent the same candidate solution in the next iteration.

(iii) The synthesizer receives the updated set $\mathcal{E} = \{R(1, 0)\}$ of examples, finds a ranking function $f(x) = x$ for $\mathcal{E}$ (i.e. for the transition from $x = 1$ to $x' = 0$), and returns a candidate solution $R(x, x') = x > x' \land x \geq 0$.

(iv) The validator checks the candidate solution, finds a counterexample $x = -2, x' = -1$ for (4.1), and adds $R(-2, -1)$ to $\mathcal{E}$.

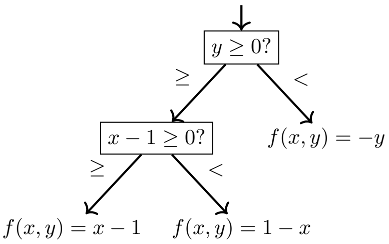
(v) The synthesizer finds a ranking function $f(x) = |x|$ for $\mathcal{E}$ and returns $R(x, x') = |x| > |x'| \land |x| \geq 0$ as a candidate solution. Note that the synthesizer have to synthesize a piecewise affine function here, but details are deferred to Section 4.1.2.

(vi) The validator accepts the candidate solution because it is a genuine solution for $\mathcal{C}$.

4.1.2 Handling Cycles in Decision Tree Learning

We explain the importance of handling cycles in our decision tree-based synthesizer of piecewise affine ranking functions.

In what follows, we deal with such decision trees as shown in Fig. 4.2: their internal nodes have affine inequalities (i.e. halfspaces); their leaves have affine functions; and overall, such a decision tree expresses a piecewise affine function



$$f(x, y) = \begin{cases} x-1 & y \geq 0 \wedge x-1 \geq 0 \\ 1-x & y \geq 0 \wedge x-1 < 0 \\ -y & y < 0 \end{cases}$$

Figure 4.2: An example of a decision tree that represents a piecewise affine ranking function $f(x, y)$

(Fig. 4.2). When we remove leaf labels from such a decision tree, then we obtain a template of piecewise functions where condition guards are given but function bodies are not. We shall call the latter a *segmentation.*

Input and output of our synthesizer. The input of our synthesizer is a set $\mathcal{E}$ of transition examples (e.g. $\mathcal{E} = \{R(1,0), R(-2,-1)\}$) as explained in Section 4.1.1. The output of our synthesizer is a well-founded relation $R(\widetilde{x}, \widetilde{x}') \coloneqq f(\widetilde{x}) > f(\widetilde{x}') \wedge f(\widetilde{x}) \geq 0$ where $\widetilde{x}$ is a sequence of variables and $f(\widetilde{x})$ is a piecewise affine function, which is represented by a decision tree (Fig. 4.2). Therefore our synthesizer aims at *learning* a suitable decision tree.

Refining segmentations and handling cycles. Roughly speaking, our synthesizer learns decision trees in the following steps.

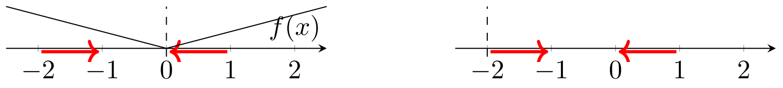
1. Generate a set H of halfspaces from the given set $\mathcal{E}$ of examples. This H serves as the vocabulary for internal nodes. Set the initial segmentation to be the one-node tree (i.e. the trivial segmentation).

2. Try to synthesize a piecewise affine ranking function f for $\mathcal{E}$ with the current segmentation—that is, try to find suitable leaf labels. If found, then use this f in a candidate well-founded relation $R(\widetilde{x}, \widetilde{x}') = f(\widetilde{x}) > f(\widetilde{x}') \wedge f(\widetilde{x}) \geq 0$.

3. Otherwise, refine the current segmentation with some halfspace in H, and go to Step 2.

The key step of our synthesizer is Step 3. We show a few examples.

Example 4.1.3. Suppose we are given $\mathcal{E} = \{R(1,0), R(-2,-1)\}$ as a set of examples. Our synthesizer proceeds as follows: (1) Our synthesizer generates the set $H \coloneqq \{x \geq 1, x \geq 0, x \geq -2, x \geq -1\}$ from the examples in $\mathcal{E}$. (2) Our synthesizer tries to find a ranking function of the form $f(x) = ax + b$ (with the trivial segmentation), but there is no such ranking function. (3) Our synthesizer refines the current segmentation with $(x \geq 0) \in H$ because $x \geq 0$ "looks good". (4) Our synthesizer tries to find a ranking function of the form $f(x) = \textbf{if } x \geq 0 \textbf{ then } ax + b \textbf{ else } cx + d$, using the current segmentation. Our synthesizer obtains $f(x) = \textbf{if } x \geq 0 \textbf{ then } x \textbf{ else } -x$ and use this $f(x)$ for a candidate solution.



(a) Good ($x \geq 0$)

(b) Bad ($x \geq -2$)

Figure 4.3: Selecting halfspaces. Transition examples are shown by red arrows. Boundaries of halfspaces are shown by dashed lines.

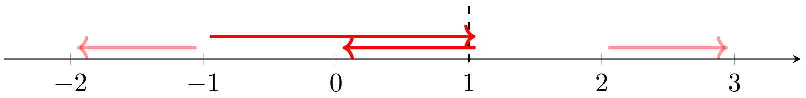


Figure 4.4: Two examples $R(-1, 1)$ and $R(1, 0)$ make an implicit cycle between $x \geq 1$ and $\neg(x \geq 1)$.

How can we decide which halfspace in H "looks good"? We use *quality measure* that is a value representing the quality of each halfspace and select the halfspace with the maximum quality measure.

Fig. 4.3 shows the comparison of the quality of $x \geq 0$ and $x \geq -2$ in this example. Intuitively, $x \geq 0$ is better than $x \geq -2$ because we can obtain a simple ranking function **if** $x \geq 0$ **then** x **else** $-x$ with $x \geq 0$ (Fig. 4.3a) while we need further refinement of the segmentation with $x \geq -2$ (Fig. 4.3b). In Section 4.4, we introduce a quality measure for halfspaces following this intuition.

Our synthesizer iteratively refines segmentations following this quality measure, until examples contained in each leaf of the decision tree admit an affine ranking function. This approach is inspired by the use of information gain in the decision tree learning for invariant synthesis.

Example 4.1.3 showed a natural extension of a decision tree learning method for invariant synthesis. However, this is not enough for transition examples, for the reasons of *explicit* and *implicit cycles*. Here are their examples.

Example 4.1.4. Suppose we are given $\mathcal{E} = \{R(1, 0), R(0, 1)\}$. In this case, there is no ranking function because $\mathcal{E}$ contains a cycle $1 \to 0 \to 1$ witnessing non-termination. We call such a cycle an *explicit cycle*.

Example 4.1.5. Let $\mathcal{E} = \{R(-1, 1), R(1, 0), R(-1, -2), R(2, 3)\}$ (Fig. 4.4). Our synthesizer proceeds as follows. (1) Our synthesizer generates the set $H := \{x \geq 1, x \geq 0, \dots\}$ of halfspaces. (2) Our synthesizer tries to find a ranking function of the form $f(x) = ax + b$ (with the trivial segmentation), but there is no such. (3) Our synthesizer refines the current segmentation with $(x \geq 1) \in H$ because $x \geq 1$ "looks good" (i.e. is the best with respect to a quality measure).

We have reached the point where the naive extension of decision tree learning explained in Example 4.1.3 no longer works: although all constraints contained in each leaf of the decision tree admit an affine ranking function, there is no piecewise affine ranking function for $\mathcal{E}$ of the form $f(x) =$ **if** $x \geq 1$ **then** $ax + b$ **else** $cx + d$.

More specifically, in this example, the leaf representing $x \geq 1$ contains $R(2, 3)$, and the other leaf representing $\neg(x \geq 1)$ contains $R(-1, -2)$. The example $R(2, 3)$ admits an affine ranking function $f_1(x) = -x + 2$, and $R(-1, -2)$

admits $f_2(x) = x + 1$, respectively. However, the combination $f(x) = \textbf{if } x \geq 1 \textbf{ then } f_1(x) \textbf{ else } f_2(x)$ is not a ranking function for $\mathcal{E}$. Moreover, there is no ranking function for $\mathcal{E}$ of the form $f(x) = \textbf{if } x \geq 1 \textbf{ then } ax + b \textbf{ else } cx + d$.

It is clear that this failure is caused by the *crossing examples* $R(-1, 1)$ and $R(1, 0)$. It is not that every crossing example is harmful. However, in this case, the set $\{R(-1, 1), R(1, 0)\}$ forms a cycle between the leaf for $x \geq 1$ and the leaf for $\neg(x \geq 1)$ (see Fig. 4.4). This "cycle" among leaves—in contrast to *explicit* cycles such as $\{R(1, 0), R(0, 1)\}$ in Example 4.1.4—is called an *implicit cycle*.

Once an implicit cycle is found, our synthesizer cuts it by refining the current segmentation. Our synthesizer continues the above steps (1–3) of decision tree learning as follows. (4) Our synthesizer selects $(x \geq 0) \in H$ and cuts the implicit cycle $\{R(-1, 1), R(1, 0)\}$ by refining segmentations. (5) Using the refined segmentation, our synthesizer obtains $f(x) = \textbf{if } x \geq 1 \textbf{ then } -x+2 \textbf{ else if } x \geq 0 \textbf{ then } 0 \textbf{ else } x + 3$ as a ranking function for $\mathcal{E}$.

As explained in Example 4.1.4,4.1.5, handling (explicit and implicit) cycles is crucial in decision tree learning for transition examples. Moreover, our *cycle detection theorem* (Theorem 4.4.5) claims that if there is no explicit or implicit cycle, then one can find a ranking function for $\mathcal{E}$ without further refinement of segmentations.

4.2 (Non-)Termination Verification as Constraint Solving

We explain how to encode (non-)termination verification to constraint solving.

Following [118], we formalize our target class pwCSP of predicate constraint satisfaction problems parametrized by a first-order theory $\mathcal{T}$.

Definition 4.2.1. Given a formula ϕ, let *ftv*(ϕ) be the set of free term variables and *fpv*(ϕ) be the set of free predicate variables in ϕ.

Definition 4.2.2. A pwCSP is defined as a pair $(\mathcal{C}, \mathcal{R})$ where $\mathcal{C}$ is a finite set of clauses of the form

$$\phi \vee \left(\bigvee_{i=1}^{\ell} X_i(\widetilde{t_i}) \right) \vee \left(\bigvee_{i=\ell+1}^{m} \neg X_i(\widetilde{t_i}) \right) \tag{4.3}$$

and $\mathcal{R} \subseteq$ *fpv*$(\mathcal{C})$ is a set of predicate variables that are required to denote *well-founded* relations. Here, $0 \leq \ell \leq m$. Meta-variables t and ϕ range over $\mathcal{T}$-terms and $\mathcal{T}$-formulas, respectively, such that *ftv*$(\phi) = \emptyset$. Meta-variables x and X range over term and predicate variables, respectively.

A pwCSP $(\mathcal{C}, \mathcal{R})$ is called CHCs (constrained Horn clauses, [20]) if $\mathcal{R} = \emptyset$ and $\ell \leq 1$ for all clauses $c \in \mathcal{C}$. The class of CHCs has been widely studied in the verification community [25, 36, 98, 127].

Definition 4.2.3. A *predicate substitution* σ is a finite map from predicate variables X to closed predicates of the form $\lambda x_1, \dots, x_{\mathrm{ar}(X)}.\phi$. We write $\mathrm{dom}(\sigma)$ for the domain of σ and $\sigma(\mathcal{C})$ for the application of σ to $\mathcal{C}$.

Definition 4.2.4. A predicate substitution σ is a *(genuine) solution* for $(\mathcal{C}, \mathcal{R})$ if (1) $fpv(\mathcal{C}) \subseteq \mathrm{dom}(\sigma)$; (2) $\models \bigwedge \sigma(\mathcal{C})$ holds; and (3) for all $X \in \mathcal{R}$, $\sigma(X)$ represents a well-founded relation, that is, $\mathrm{sort}(\sigma(X)) = (\widetilde{s}, \widetilde{s}) \to \bullet$ for some sequence $\widetilde{s}$ of sorts and there is no infinite sequence $\widetilde{v}_1, \widetilde{v}_2, \ldots$ of sequences $\widetilde{v}_i$ of values of the sorts $\widetilde{s}$ such that $\models \rho(X)(\widetilde{v}_i, \widetilde{v}_{i+1})$ for all $i \geq 1$.

Encoding termination. Given a set of initial states $\iota(\widetilde{x})$ and a transition relation $\tau(\widetilde{x}, \widetilde{x}')$, the termination verification problem is expressed by the pwCSP $(\mathcal{C}, \mathcal{R})$ where $\mathcal{R} = \{R\}$, and $\mathcal{C}$ consists of the following clauses.

$$\iota(\widetilde{x}) \implies I(\widetilde{x}) \qquad \tau(\widetilde{x}, \widetilde{x}') \land I(\widetilde{x}) \implies I(\widetilde{x}') \qquad \tau(\widetilde{x}, \widetilde{x}') \land I(\widetilde{x}) \implies R(\widetilde{x}, \widetilde{x}')$$

We use $\phi \implies \psi$ as syntax sugar for $\neg\phi \lor \psi$, so this is a pwCSP. The well-founded relation R asserts that τ is terminating. We also consider an invariant I for τ to avoid synthesizing ranking functions on unreachable program states.

Encoding non-termination. We can encode a problem of non-termination verification to a pwCSP via recurrent sets [47]. The existence of a recurrent set witnesses an infinite run of the program. For simplicity of explanation, we assume there is only a single loop with one program variable where $\iota(x)$ is a set of initial states, $\tau(x, x')$ is a transition relation of the loop body, and $\gamma(x)$ is a guard condition. A recurrent set is a set R of program states satisfying the following condition.

1. The set R implies the guard condition γ.
2. Some state reachable from ι satisfies R.
3. For any state x in R, there exists a successor state x' in R.

These conditions can be expressed as follows.

$$\begin{aligned} R(x) &\implies \gamma(x) \\ &\exists x.\iota(x) \land R(x) && (4.4) \\ R(x) &\implies \exists x'.\tau(x, x') \land R(x') && (4.5) \end{aligned}$$

We need to remove $\exists$ from (4.4) and (4.5) because such occurrences of $\exists$ are not allowed in pwCSP. The existential quantifier $\exists$ in (4.5) can be removed by considering the following constraint where S is a well-founded relation.

$$\begin{aligned} R(x) &\implies E(x, 0) && (4.6) \\ E(x, x') &\implies \big(\tau(x, x') \land R(x')\big) \\ &\qquad \lor \big(S(x', x'-1) \land E(x, x'-1)\big) \lor \big(S(x', x'+1) \land E(x, x'+1)\big) && (4.7) \end{aligned}$$

The intuition is as follows. Given x in the recurrent set R, the relation $E(x, x')$ searches for the value of $\exists x'$ in (4.5). The search starts from $x' = 0$ in (4.6), and x' is nondeterministically incremented or decremented in (4.7). The well-founded relation S asserts that the search finishes within finite steps. The existential quantifier $\exists$ in (4.4) is removed in the same way as (4.5). As a result, we obtain a pwCSP for the non-termination problem.

Example 4.2.5. Consider the following C program.

```
while(x > 0) { x = -2 * x + 9; }
```

The non-termination problem is encoded as the pwCSP $(\mathcal{C}, \mathcal{R})$ where $\mathcal{R} = \{S, S'\}$, and $\mathcal{C}$ consists of

$$\begin{aligned}
R(x) &\implies x > 0 \\
\top &\implies E(0) \\
E(x) &\implies R(x) \vee (S(x, x-1) \wedge E(x-1)) \vee (S(x, x+1) \wedge E(x+1)) \\
R(x) &\implies E'(x, 0) \\
E'(x, x') &\implies (x' = -2x + 9 \wedge R(x')) \\
&\quad\ \vee (S'(x', x'-1) \wedge E'(x, x'-1)) \\
&\quad\ \vee (S'(x', x'+1) \wedge E'(x, x'+1)).
\end{aligned}$$

The program is non-terminating when $x = 3$. This is witnessed by the solution σ for $(\mathcal{C}, \mathcal{R})$ defined as follows.

$$\begin{aligned}
\sigma(R)(x) &\quad := \quad x = 3 \\
\sigma(E)(x) &\quad := \quad 0 \leq x \wedge x \leq 3 \\
\sigma(S)(x, x') &\quad := \quad x' = x + 1 \wedge x' \leq 3 \\
\sigma(E')(x, x') &\quad := \quad x = 3 \wedge 0 \leq x' \wedge x' \leq 3 \\
\sigma(S')(x, x') &\quad := \quad x' = x + 1 \wedge x' \leq 3
\end{aligned}$$

In the general case where there are multiple (possibly nested) loops with many program variables, we can generalize the above encoding by characterizing the set of terminating states as a least fixed point, taking the dual to obtain the set of non-terminating states, and removing the existential quantifiers.

4.3 CounterExample-Guided Inductive Synthesis (CEGIS)

We explain how CounterExample-Guided Inductive Synthesis [107] (CEGIS for short) works for a given pwCSP $(\mathcal{C}, \mathcal{R})$ following [118]. Then, we add the extraction of positive/negative examples to the CEGIS architecture, which enables our decision tree-based synthesizer to use a simplified form of examples.

CEGIS proceeds through the iterative interaction between a synthesizer and a validator (Fig. 1.5), in which they exchange examples and candidate solutions.

Definition 4.3.1. A formula ϕ is an *example* of $\mathcal{C}$ if $\mathit{ftv}(\phi) = \emptyset$ and $\bigwedge \mathcal{C} \models \phi$ hold. Given a set $\mathcal{E}$ of examples of $\mathcal{C}$, a predicate substitution σ is a *candidate solution* for $(\mathcal{C}, \mathcal{R})$ that is consistent with $\mathcal{E}$ if σ is a solution for $(\mathcal{E}, \mathcal{R})$.

Synthesizer. The input for a synthesizer is a set $\mathcal{E}$ of examples of $\mathcal{C}$ collected from previous CEGIS iterations. The synthesizer tries to find a candidate solution σ consistent with $\mathcal{E}$ instead of a genuine solution for $(\mathcal{C}, \mathcal{R})$. If the candidate solution σ is found, then σ is passed to the validator. If $\mathcal{E}$ is unsatisfiable, then $\mathcal{E}$ witnesses unsatisfiability of $(\mathcal{C}, \mathcal{R})$. Details of our synthesizer is described in Section 4.4.

Validator. A validator checks whether the candidate solution σ from the synthesizer is a genuine solution of $(\mathcal{C}, \mathcal{R})$ by using SMT solvers. That is, satisfiability of $\models \neg \bigwedge \sigma(\mathcal{C})$ is checked. If $\models \neg \bigwedge \sigma(\mathcal{C})$ is not satisfiable, then σ is a genuine solution of the original pwCSP $(\mathcal{C}, \mathcal{R})$, so the validator accepts this. Otherwise, the validator adds new examples to the set $\mathcal{E}$ of examples. Finally, the synthesizer is invoked again with the updated set $\mathcal{E}$ of examples.

If $\models \neg \bigwedge \sigma(\mathcal{C})$ is satisfiable, new examples are constructed as follows. Using SMT solvers, the validator obtains an assignment θ to term variables such that $\models \neg\theta(\psi)$ holds for some $\psi \in \sigma(\mathcal{C})$. By (4.3), $\models \neg\theta(\psi)$ is a clause of the form $\models \neg\theta(\phi) \wedge \big(\bigwedge_{i=1}^{\ell} \neg\sigma(X_i)(\theta(\widetilde{t_i}))\big) \wedge \big(\bigwedge_{i=\ell+1}^{m} \sigma(X_i)(\theta(\widetilde{t_i}))\big)$. To prevent this counterexample from being found in the next CEGIS iteration again, the validator adds the following example to $\mathcal{E}$.

$$\bigvee_{i=1}^{\ell} X_i(\theta(\widetilde{t_i})) \vee \bigvee_{i=\ell+1}^{m} \neg X_i(\theta(\widetilde{t_i})) \tag{4.8}$$

The CEGIS architecture repeats this interaction between the synthesizer and the validator until a genuine solution for $(\mathcal{C}, \mathcal{R})$ is found or $\mathcal{E}$ witnesses unsatisfiability of $(\mathcal{C}, \mathcal{R})$.

Extraction of positive/negative examples. Examples obtained in the above explanation are a bit complex to handle in our decision tree-based synthesizer: each example in $\mathcal{E}$ is a disjunction (4.8) of literals, which may contain multiple predicate variables.

To simplify the form of examples, we extract from $\mathcal{E}$ the sets $\mathcal{E}_X^+$ and $\mathcal{E}_X^-$ of *positive examples* (i.e., examples of the form $X(\widetilde{v})$) and *negative examples* (i.e., examples of the form $\neg X(\widetilde{v})$) for each $X \in \mathit{fpv}(\mathcal{E})$. This allows us to synthesize a predicate $\sigma(X)$ for each predicate variable $X \in \mathit{fpv}(\mathcal{E})$ separately. For simplicity, we write $\widetilde{v} \in \mathcal{E}_X^+$ and $\widetilde{v} \in \mathcal{E}_X^-$ instead of $X(\widetilde{v}) \in \mathcal{E}_X^+$ and $\neg X(\widetilde{v}) \in \mathcal{E}_X^-$.

The extraction is done as follows. We first substitute for each predicate variable application $X(\widetilde{v})$ in $\mathcal{E}$ a boolean variable $b_{X(\widetilde{v})}$ to obtain a SAT problem $\mathbf{SAT}(\mathcal{E})$. Then, we use SAT solvers to obtain an assignment η that is a solution for $\mathbf{SAT}(\mathcal{E})$. If a solution η exists, then we construct positive/negative examples from η; otherwise, $\mathcal{E}$ is unsatisfiable.

Definition 4.3.2. Let η be a solution for $\mathbf{SAT}(\mathcal{E})$. For each predicate variable $X \in \mathit{fpv}(\mathcal{E})$, we define the set $\mathcal{E}_X^+$ of *positive examples* and the set $\mathcal{E}_X^+$ of *negative examples* under the assignment η by $\mathcal{E}_X^+ := \{\widetilde{v} \mid \eta(b_{X(\widetilde{v})}) = \mathbf{true}\}$ and $\mathcal{E}_X^- := \{\widetilde{v} \mid \eta(b_{X(\widetilde{v})}) = \mathbf{false}\}$.

Note that some of predicate variable applications $X(\widetilde{v})$ may not be assigned true nor false because they do not affect the evaluation of $\mathbf{SAT}(\mathcal{E})$. Such predicate variable applications are discarded from $\{(\mathcal{E}_X^+, \mathcal{E}_X^-)\}_{X \in \mathit{fpv}(\mathcal{E})}$.

Our method uses the extraction of positive and negative examples when the validator passes examples to the synthesizer. If $X \in \mathit{fpv}(\mathcal{E}) \cap \mathcal{R}$, then we apply our ranking function synthesizer to $(\mathcal{E}_X^+, \mathcal{E}_X^-)$. If $X \in \mathit{fpv}(\mathcal{E}) \setminus \mathcal{R}$, then we apply an invariant synthesizer.

We say a candidate solution σ is consistent with $\{(\mathcal{E}_X^+, \mathcal{E}_X^-)\}_{X \in \mathit{fpv}(\mathcal{E})}$ if $\models \sigma(X)(\widetilde{v}^+)$ and $\models \neg\sigma(X)(\widetilde{v}^-)$ hold for each predicate variable $X \in \mathit{fpv}(\mathcal{E})$, $\widetilde{v}^+ \in \mathcal{E}_X^+$, and $\widetilde{v}^- \in \mathcal{E}_X^-$. If a candidate solution σ is consistent with $\{(\mathcal{E}_X^+, \mathcal{E}_X^-)\}_{X \in \mathit{fpv}(\mathcal{E})}$, then σ is also consistent with $\mathcal{E}$.

Note that unsatisfiability of $\{(\mathcal{E}_X^+, \mathcal{E}_X^-)\}_{X \in fpv(\mathcal{E})}$ does not immediately implies unsatisfiability of $\mathcal{E}$ nor $(\mathcal{C}, \mathcal{R})$ because $\{(\mathcal{E}_X^+, \mathcal{E}_X^-)\}_{X \in fpv(\mathcal{E})}$ depends on the choice of the assignment η. Therefore, the CEGIS architecture need to be modified: if synthesizers find unsatisfiability of $\{(\mathcal{E}_X^+, \mathcal{E}_X^-)\}_{X \in fpv(\mathcal{E})}$, then we add the negation of an unsatisfiability core to $\mathcal{E}$ to prevent using the same assignment η again.

Note that some restricted forms of (4.8) have also been considered in previous work and are called implication examples in [43] and implication/negation constraints in [25]. Our extraction of positive and negative examples is applicable to the general form of (4.8).

4.4 Ranking Function Synthesis

In this section, we describe one of the main contributions, that is, our decision tree-based synthesizer, which synthesizes a candidate well-founded relation $\sigma(R)$ from a finite set $\mathcal{E}_R^+$ of examples. We assume that only positive examples are given because well-founded relations occur only positively in pwCSP for termination analysis (see Section 4.2). The aim of our synthesizer is to find a piecewise affine lexicographic ranking function $\widetilde{f}(\widetilde{x})$ for the given set $\mathcal{E}_R^+$ of examples. Below, we fix a predicate variable $R \in \mathcal{R}$ and omit the subscript $\mathcal{E}_R^+ = \mathcal{E}^+$.

4.4.1 Basic Definitions

To represent piecewise affine lexicographic ranking functions, we use decision trees like the one in Figure 4.2. Let $\widetilde{x} = (x_1, \dots, x_n)$ be the program variables where each x_i ranges over $\mathbb{Z}$.

Definition 4.4.1. A *decision tree* D is defined by $D \coloneqq \widetilde{g}(\widetilde{x}) \mid \textbf{if } h(\widetilde{x}) \geq 0 \textbf{ then } D \textbf{ else } D$ where $\widetilde{g}(\widetilde{x}) = (g_k(\widetilde{x}), \dots, g_0(\widetilde{x}))$ is a tuple of affine functions and $h(\widetilde{x})$ is an affine function. A *segmentation tree* S is defined as a decision tree with undefined leaves $\bot$: that is, $S \coloneqq \bot \mid \textbf{if } h(\widetilde{x}) \geq 0 \textbf{ then } S \textbf{ else } S$. For each decision tree D, we can canonically assign a segmentation tree by replacing the label of each leaf with $\bot$. This is denoted by $S(D)$. For each decision tree D, we denote the corresponding piecewise affine function by $\widetilde{f}_D(\widetilde{x}) : \mathbb{Z}^n \to \mathbb{Z}^{k+1}$.

Each leaf in a segmentation tree S corresponds to a polyhedron. We often identify the segmentation tree S with the set of leaves of S and a leaf with the polyhedron corresponding to the leaf. For example, we say something like "for each $L \in S$, $\widetilde{v} \in L$ is a point in the polyhedron L".

Suppose we are given a segmentation tree S and a set $\mathcal{E}^+$ of examples.

Definition 4.4.2. For each $L_1, L_2 \in S$, we denote the set of example transitions from L_1 to L_2 by $\mathcal{E}_{L_1,L_2}^+ \coloneqq \{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+ \mid \widetilde{v} \in L_1, \widetilde{v}' \in L_2\}$. An example $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+$ is *crossing* w.r.t. S if $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}_{L_1,L_2}^+$ for some $L_1 \neq L_2$, and *non-crossing* if $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}_{L,L}^+$ for some L.

Definition 4.4.3. We define the *dependency graph* $G(S, \mathcal{E}^+)$ for S and $\mathcal{E}^+$ by the graph (V, E) where vertices $V = S$ are leaves, and edges $E = \{(L_1, L_2) \mid L_1 \neq L_2, \exists (\widetilde{v}, \widetilde{v}') \in \mathcal{E}_{L_1,L_2}^+\}$ are crossing examples.

We denote the set of start points $\widetilde{v}$ and end points $\widetilde{v}'$ of examples $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+$ by $\underline{\mathcal{E}^+} := \{\widetilde{v} \mid (\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+\} \cup \{\widetilde{v}' \mid (\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+\}$.

4.4.2 Segmentation and (Explicit and Implicit) Cycles: One-Dimensional Case

For simplicity, we first consider the case where $\widetilde{f}(\widetilde{x}) = f(\widetilde{x}) : \mathbb{Z}^n \to \mathbb{Z}$ is a one-dimensional ranking function. Our aim is to find a ranking function $f(\widetilde{x})$ for $\mathcal{E}^+$, which satisfies $\forall(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+.\ f(\widetilde{v}) > f(\widetilde{v}')$ and $\forall(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+.\ f(\widetilde{v}) \geq 0$. If our ranking function synthesizer finds such a ranking function $f(\widetilde{x})$, then a candidate well-founded relation R_f is constructed as $R_f(\widetilde{x}, \widetilde{x}') := f(\widetilde{x}) \geq 0 \wedge f(\widetilde{x}) > f(\widetilde{x}')$.

Our synthesizer builds a decision tree D to find a ranking function $f_D(\widetilde{x})$ for $\mathcal{E}^+$. The main question in doing so is "when and how should we refine partitions of decision trees?" To answer this question, we consider the case where there is no ranking function $f_D(\widetilde{x})$ for $\mathcal{E}^+$ with a fixed segmentation S, and classify reasons for this into three cases as follows.

Case 1: explicit cycles in examples. We define an *explicit cycle* in $\mathcal{E}^+$ as a cycle in the graph $(\mathbb{Z}^n, \mathcal{E}^+)$. An explicit cycle witnesses that there is no ranking function for $\mathcal{E}^+$ (see e.g., Example 4.1.4).

Case 2: non-crossing examples are unsatisfiable. The second case is when there is a leaf $L \in S$ such that no affine (not *piecewise* affine) ranking function for the set $\mathcal{E}^+_{L,L}$ of non-crossing examples exists. This prohibits the existence of piecewise affine function $f_D(\widetilde{x})$ for $\mathcal{E}^+$ with segmentation $S = S(D)$ because the restriction of $f_D(\widetilde{x})$ to $L \in S$ must be an affine ranking function for $\mathcal{E}^+_{L,L}$.

Case 3: implicit cycles in the dependency graph. We define an *implicit cycle* by a cycle in the dependency graph $G(S, \mathcal{E}^+)$. Case 3 is the case where an implicit cycle prohibits the existence of piecewise affine ranking functions for $\mathcal{E}^+$ with the segmentation S (e.g., Example 4.1.5). If Case 1 and Case 2 do not hold but no piecewise affine ranking function for $\mathcal{E}^+$ with the segmentation S exists, then there must be an implicit cycle by (the contraposition of) the following proposition.

Proposition 4.4.4. Assume $\mathcal{E}^+$ is a set of examples that does not contain explicit cycles (i.e. Case 1 does not hold). Let S be a segmentation tree and assume that for each $L \in S$, there exists an affine ranking function $f_L(\widetilde{x})$ for $\mathcal{E}^+_{L,L}$ (i.e. Case 2 does not hold). If the dependency graph $G(S, \mathcal{E}^+)$ is acyclic, then there exists a decision tree D with the segmentation $S(D) = S$ such that $f_D(\widetilde{x})$ is a ranking function for $\mathcal{E}^+$.

Proof. By induction on the height (i.e. the length of a longest path from a vertex) of vertices in $G(S, \mathcal{E}^+)$. We construct a decision tree D as follows. If the height of $L \in S$ is 0, then we assign $f'_L(\widetilde{x}) := f_L(\widetilde{x})$ to the leaf L where $f_L(\widetilde{x})$ is a ranking function for $\mathcal{E}^+_{L,L}$. If the height of $L \in S$ is $n > 0$, then we assign $f'_L(\widetilde{x}) := f_L(\widetilde{x}) + c$ to the leaf L where $c \in \mathbb{Z}$ is a constant that satisfies $\forall(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+_{L,L'}, f_L(\widetilde{v}) + c > f'_{L'}(\widetilde{v}')$ for each cell L' with the height less than n. □ □

Note that the converse of Proposition 4.4.4 does not hold: the existence of implicit cycles in $G(S, \mathcal{E}^+)$ does not necessarily imply that no piecewise affine ranking function exists with the segmentation S.

4.4.3 Segmentation and (Explicit and Implicit) Cycles: Multi-Dimensional Lexicographic Case

We consider a more general case where $\widetilde{f}(\widetilde{x}) = (f_k(\widetilde{x}), \ldots, f_0(\widetilde{x}))$ is a multi-dimensional lexicographic ranking function and k is a fixed nonnegative integer.

Given a function $\widetilde{f}(\widetilde{x})$, we consider the well-founded relation $R_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ defined inductively as follows.

$$R_{()}(\widetilde{x}, \widetilde{x}') := \bot \qquad R_{(f_k,\ldots,f_0)}(\widetilde{x}, \widetilde{x}') \; := f_k(\widetilde{x}) \geq 0 \wedge f_k(\widetilde{x}) > f_k(\widetilde{x}') \\ \vee f_k(\widetilde{x}) = f_k(\widetilde{x}') \wedge R_{(f_{k-1},\ldots,f_0)}(\widetilde{x}, \widetilde{x}') \tag{4.9}$$

Our aim here is to find a lexicographic ranking function $\widetilde{f}(\widetilde{x})$ for $\mathcal{E}^+$, i.e. a function $\widetilde{f}(\widetilde{x})$ such that $R_{\widetilde{f}}(\widetilde{v}, \widetilde{v}')$ holds for each $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+$. Our synthesizer does so by building a decision tree. The same argument as the one-dimensional case holds for lexicographic ranking functions.

Theorem 4.4.5 (cycle detection)**.** Assume $\mathcal{E}^+$ is a set of examples that does not contain explicit cycles. Let S be a segmentation tree and assume that for each $L \in S$, there exists an affine function $\widetilde{f}_L(\widetilde{x})$ that satisfies $\forall(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+_{L,L}, R_{\widetilde{f}_L}(\widetilde{v}, \widetilde{v}')$. If the dependency graph $G(S, \mathcal{E}^+)$ is acyclic, then there exists a decision tree D with the segmentation $S(D) = S$ such that $R_{\widetilde{f}_D}(\widetilde{v}, \widetilde{v}')$ holds for each $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+$.

Proof. The proof is almost the same as Proposition 4.4.4. Here, note that if $\widetilde{f}'(\widetilde{x}) = \widetilde{f}(\widetilde{x}) + \widetilde{c}$ where $\widetilde{c}$ is a tuple of nonnegative integer constants, then $R_{\widetilde{f}'}(\widetilde{x}, \widetilde{x}')$ subsumes $R_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$. □ □

4.4.4 Our Decision Tree Learning Algorithm

We design a concrete algorithm based on Theorem 4.4.5. It is shown in Algorithm 1 and consists of three phases. We shall describe the three phases one by one.

Phase 1

Phase 1 (Line 1-3) detects explicit cycles in $\mathcal{E}^+$ to exclude Case 1. Here, we use a cycle detection algorithm for directed graphs.

Phase 2

Phase 2 (Line 4) detects and resolves Case 2 by using ResolveCase2 (Algorithm 2), which is a function that grows a decision tree recursively. ResolveCase2 takes non-crossing examples in a leaf, divides the leaf, and returns a *template tree* that is fine enough to avoid Case 2. Here, template trees are decision trees whose leaves are labeled by affine templates.

Algorithm 2 shows the detail of ResolveCase2. ResolveCase2 builds a template tree recursively starting from the trivial segmentation $S = \bot$ and all given examples. In each polyhedron, ResolveCase2 checks whether the set C

Algorithm 1 Building decision trees.

```
Input: a set $\mathcal{E}^+$ of examples, an integer $k \geq 0$
Output: a well-founded relation $R$ such that $\forall(\widetilde{x}, \widetilde{x}') \in \mathcal{E}^+, R(\widetilde{x}, \widetilde{x}')$
if $E$ has a cycle then
    return unsatisfiable
end if
$D :=$ ResolveCase2($E$)
while true do
    $C :=$ GetConstraints($D, E$)
    $O :=$ SumAbsParams($D$)
    $\rho :=$ Minimize($O, C$)
    if $\rho$ is defined then
        $\widetilde{f}(\widetilde{x}) := \widetilde{f}_{\rho(D)}(\widetilde{x})$
        return $R_{\widetilde{f}}$
    else
        get an unsat core in $C$
        find an implicit cycle $(\widetilde{v}_1, \widetilde{v}'_1), \ldots, (\widetilde{v}_l, \widetilde{v}'_l)$ in the unsat core
        find a cell $C$ and two distinct points $\widetilde{v}'_i, \widetilde{v}_{i+1} \in C$ in the implicit cycle
        add a halfspace to separate $\widetilde{v}'_i$ and $\widetilde{v}_{i+1}$ and update $D$
    end if
end while
```

of constraints imposed by non-crossing examples can be satisfied by an affine lexicographic ranking function on the polyhedron (Line 2-3). If the set C of constraints is not satisfiable, then ResolveCase2 chooses a halfspace $h(\widetilde{x}) \geq 0$ (Line 6) and divides the current polyhedron by the halfspace.

There is a certain amount of freedom in the choice of halfspaces. To guarantee termination of the whole algorithm, we require that the chosen halfspace h separates at least one point in $\underline{\mathcal{E}'^+} := \{\widetilde{v} \mid (\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^+\} \cup \{\widetilde{v}' \mid (\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^+\}$ from the other points in $\underline{\mathcal{E}'^+}$. That is:

Assumption 4.4.6. If halfspace $h(\widetilde{x}) \geq 0$ is chosen in Line 6 of Algorithm 2, then there exist $\widetilde{v}, \widetilde{u} \in \underline{\mathcal{E}'^+}$ such that $h(\widetilde{v}) \geq 0$ and $h(\widetilde{u}) < 0$.

We explain two strategies (eager and lazy) to choose halfspaces that can be used to implement ChooseQualifier. Both of them are guaranteed to terminate, and moreover, intended to yield simple decision trees.

Eager strategy. In the eager strategy, we eagerly generate a finite set H of halfspaces from the set $\mathcal{E}^+$ of all examples beforehand and choose the best one from H with respect to a certain quality measure. To satisfy Assumption 4.4.6, H are generated so that any two points $\widetilde{u}, \widetilde{v} \in \underline{\mathcal{E}^+}$ can be separated by some halfspace $(h(\widetilde{x}) \geq 0) \in H$.

For example, we can use intervals $H = \{\pm(x_i - a_i) \geq 0 \mid i = 1, \ldots, n \wedge (a_1, \ldots, a_n) \in \underline{\mathcal{E}^+}\}$ and octagons $H = \{\pm(x_i - a_i) \pm (x_j - a_j) \geq 0 \mid i \neq j \wedge (a_1, \ldots, a_n) \in \underline{\mathcal{E}^+}\}$ where $\widetilde{x} = (x_1, \ldots, x_n)$. For any input $\mathcal{E}'^+ \subseteq \mathcal{E}^+$ of ResolveCase2, intervals and octagons satisfy $\emptyset \neq H' := \{h(\widetilde{x}) \geq 0 \mid \exists \widetilde{v}, \widetilde{u} \in \mathcal{E}'^+ . h(\widetilde{v}) \geq 0 \wedge h(\widetilde{u}) < 0\}$, so Assumption 4.4.6 is satisfied by choosing the best halfspace with respect to the quality measure from H'.

Algorithm 2 Resolving Case 2.

```
function ResolveCase2($\mathcal{E}'^{+}$)
    $\widetilde{f} :=$ MakeAffineTemplate($k$)
    $C :=$ GetConstraints($\widetilde{f}, \mathcal{E}'^{+}$)
    $\rho :=$ GetModel($C$)
    if $\rho$ is undefined then
        $h :=$ ChooseQualifier($\mathcal{E}'^{+}$)
        $D_{\geq 0} :=$ ResolveCase2($\{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^{+} \mid h(\widetilde{v}) \geq 0 \wedge h(\widetilde{v}') \geq 0\}$)
        $D_{<0} :=$ ResolveCase2($\{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^{+} \mid h(\widetilde{v}) < 0 \wedge h(\widetilde{v}') < 0\}$)
        return (if $h(\widetilde{x}) \geq 0$ then $D_{\geq 0}$ else $D_{<0}$)
    else
        return $\widetilde{f}$
    end if
end function
function GetConstraints($D, \mathcal{E}^{+}$)
    return $\{R_{\widetilde{f}_D}(\widetilde{v}, \widetilde{v}') \mid (\widetilde{v}, \widetilde{v}') \in \mathcal{E}^{+}\}$ where $\widetilde{f}_D$ is the tuple of piecewise
    affine functions corresponding to $D$
end function
```

Algorithm 3 A criterion for eager qualifier selection.

```
function QualityMeasure($h, \mathcal{E}'^{+}$)
    $E_{++} := \{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^{+} \mid h(\widetilde{v}) \geq 0 \wedge h(\widetilde{v}) \geq 0\}$
    $E_{+-} := \{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^{+} \mid h(\widetilde{v}) \geq 0 \wedge h(\widetilde{v}) < 0\}$
    $E_{-+} := \{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^{+} \mid h(\widetilde{v}) < 0 \wedge h(\widetilde{v}) \geq 0\}$
    $E_{--} := \{(\widetilde{v}, \widetilde{v}') \in \mathcal{E}'^{+} \mid h(\widetilde{v}) < 0 \wedge h(\widetilde{v}) < 0\}$
    $\widetilde{f} :=$ MakeAffineTemplate($k$)
    $C_{+} :=$ GetConstraints($\widetilde{f}, E_{++}$)
    $C_{-} :=$ GetConstraints($\widetilde{f}, E_{--}$)
    $N_{+} :=$ MaxSmt($C_{+}$)
    $N_{-} :=$ MaxSmt($C_{-}$)
    return $N_{+} + N_{-} + (|E_{+-}| + |E_{-+}|)(1 - \mathrm{entropy}(|E_{+-}|, |E_{-+}|))$
end function
```

For each halfspace $(h(\widetilde{x}) \geq 0) \in H'$, we calculate QualityMeasure in Algorithm 3, and choose one that maximizes QualityMeasure$(h, \mathcal{E}'^{+})$. QualityMeasure$(h, \mathcal{E}'^{+})$ calculates the sum of the maximum number of satisfiable constraints in each leaf divided by $h(\widetilde{x}) \geq 0$ plus an additional term $(|E_{+-}| + |E_{-+}|)(1 - \mathrm{entropy}(|E_{+-}|, |E_{-+}|))$ where $\mathrm{entropy}(x, y) = -\frac{x}{x+y} \log_2 \frac{x}{x+y} - \frac{y}{x+y} \log_2 \frac{y}{x+y}$. Therefore, the term $(|E_{+-}| + |E_{-+}|)(1 - \mathrm{entropy}(|E_{+-}|, |E_{-+}|))$ is close to $|E_{+-}| + |E_{-+}|$ if almost all examples in $E_{+-} \cup E_{-+}$ cross h in the same direction and close to 0 if $|E_{+-}|$ is almost equal to $|E_{-+}|$.

Lazy strategy. In the lazy strategy, we lazily generate halfspaces. We divide the current polyhedron so that non-crossing examples in the cell point to almost the same direction.

First, we label states that occur in $\mathcal{E}^{+}_{C,C}$ as follows. We find a direction that most examples in C point to by solving the MAX-SMT $\vec{a} := \max_{\vec{a}} \big|\{(\widetilde{v}, \widetilde{v}') \in$

$\mathcal{E}^+_{C,C} \mid \vec{a} \cdot (\widetilde{v} - \widetilde{v}') > 0\}\big|$. For each $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+_{C,C}$, we label two points $\widetilde{v}, \widetilde{v}'$ with $+1$ if $\vec{a} \cdot (\widetilde{v} - \widetilde{v}') > 0$ and with -1 otherwise.

Then we apply weighted C-SVM to generate a hyperplane that separates most of the positive and negative points. To guarantee termination of Algorithm 1, we avoid "useless" hyperplanes that classify all the points by the same label. If we obtain such a useless hyperplane, then we undersample a majority class and apply C-SVM again. By undersampling suitably, we eventually get linearly separable data with at least one positive point and one negative point.

Note that since coefficients of hyperplanes extracted from C-SVM are floating point numbers, we have to approximate them by hyperplanes with rational coefficients. This is done by truncating continued fraction expansions of coefficients by a suitable length.

Phase 3

In Line 5-18 of Algorithm 1, we further refine the segmentation $S(D)$ to resolve Case 3. Once Case 2 is resolved by ResolveCase2, Case 2 never holds even after refining $S(D)$ further. This enables to separate Phases 2 and 3.

Given a template tree D, we consider the set C of constraints on parameters in D that claims $\widetilde{f}_D(\widetilde{x})$ is a ranking function for $\mathcal{E}^+$ (Line 6).

If C is satisfiable, we use an SMT solver to obtain a solution of C (i.e. an assignment ρ of integers to parameters) while minimizing the sum of absolute values of unknown parameters in D at the same time (Line 8). This minimization is intended to give a simple candidate ranking function. The solution ρ is used to instantiate the template tree D (Line 11).

If C cannot be satisfied, there must be an implicit cycle in the dependency graph $G(S(D), \mathcal{E}^+)$ by Theorem 4.4.5. The implicit cycle can be found in an unsatisfiable core of C. We refine the segmentation of D to cut the implicit cycle in Line 16. To guarantee termination, we choose a halfspace satisfying the following assumption, which is similar to Assumption 4.4.6.

Assumption 4.4.7. If halfspace $h(\widetilde{x}) \geq 0$ is chosen in Line 16 of Algorithm 1, then there exist $\widetilde{v}, \widetilde{u} \in \underline{\mathcal{E}^+}$ such that $h(\widetilde{v}) \geq 0$ and $h(\widetilde{u}) < 0$.

We have two strategy (eager and lazy) to refine the segmentation of D.

In eager strategy, we choose a halfspace $(h(\widetilde{x}) \geq 0) \in H$ that separates two distinct points $\widetilde{v}'_i$ and $\widetilde{v}_{i+1}$ in the implicit cycle. In doing so, we want to reduce the number of implicit cycles in $G(S(D), \mathcal{E}^+)$, but adding a new halfspace may introduce new implicit cycles if there exists $(\widetilde{v}, \widetilde{v}') \in \mathcal{E}^+_{C,C}$ that crosses the new border from the side of $\widetilde{v}'_i$ to the side of $\widetilde{v}_{i+1}$. Therefore, we choose a hyperplane that minimizes the number of new crossing examples.

In lazy strategy, we use an SMT solver to find a hyperplane $h(\widetilde{x}) \in H$ that separates $\widetilde{v}'_i$ and $\widetilde{v}_{i+1}$ and minimizes the number of new crossing examples.

Termination

Assumption 4.4.6 and Assumption 4.4.7 guarantees that every leaf in $S(D)$ contains at least one point in the finite set $\underline{\mathcal{E}^+}$. Because the number of leaves in $S(D)$ strictly increases after each iteration of Phase 2 and Phase 3, we eventually get a segmentation $S(D)$ where each $L \in S(D)$ contains only one point in

$\underline{\mathcal{E}^+}$ in the worst case. Since we have excluded Case 1 at the beginning, Theorem 4.4.5 guarantees the existence of ranking function with the segmentation $S(D)$. Therefore, the algorithm terminates within $|\underline{\mathcal{E}^+}|$ times of refinement.

Theorem 4.4.8. If Assumption 4.4.6 and Assumption 4.4.7 hold, then Algorithm 1 terminates. If Algorithm 1 returns a piecewise affine lexicographic function $\widetilde{f}(\widetilde{x})$, then the function satisfies $R_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ for each $(\widetilde{x}, \widetilde{x}') \in \mathcal{E}^+$ where $\mathcal{E}^+$ is the input of the algorithm. □

4.4.5 Improvement by Degenerating Negative Values

There is another way to define well-founded relation from the tuple $\widetilde{f}(\widetilde{x}) = (f_k(\widetilde{x}), \ldots, f_0(\widetilde{x}))$ of functions, that is, the well-founded relation $R'_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ defined inductively by $R'_{()}(\widetilde{x}, \widetilde{x}') \coloneqq \bot$ and $R'_{(f_k,\ldots,f_0)}(\widetilde{x}, \widetilde{x}') \coloneqq f_k(\widetilde{x}) \geq 0 \wedge f_k(\widetilde{x}) > f_k(\widetilde{x}') \vee \big(f_k(\widetilde{x}') < 0 \vee f_k(\widetilde{x}) = f_k(\widetilde{x}')\big) \wedge R'_{(f_{k-1},\ldots,f_0)}(\widetilde{x}, \widetilde{x}')$.

In this definition, we loosen the equality $f_i(\widetilde{x}) = f_i(\widetilde{x}')$ (where $i = 1, \ldots, k$) of the usual lexicographic ordering (4.9) to $f_i(\widetilde{x}') < 0 \vee f_i(\widetilde{x}) = f_i(\widetilde{x}')$. This means that once $f_i(\widetilde{x})$ becomes negative, $f_i(\widetilde{x})$ must stay negative but the value do not have to be the same, which is useful for the synthesizer to avoid complex candidate lexicographic ranking functions and thus improves the performance.

However, if we use this well-founded relation $R'_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ instead of $R_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ in (4.9), then Theorem 4.4.5 fails because $R'_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ is not necessarily subsumed by $R'_{\widetilde{f}+\widetilde{c}}$ where $\widetilde{c} = (c_k, \ldots, c_0)$ is a nonnegative constant (see the proof of Proposition 4.4.4 and Theorem 4.4.5). As a result, there is a chance that no implicit cycle can be found in line 14 of Algorithm 1. Therefore, when we use $R'_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$, we modify Algorithm 1 so that if no implicit cycle can be found in line 14, then we fall back on the former definition of $R_{\widetilde{f}}(\widetilde{x}, \widetilde{x}')$ and restart Algorithm 1.

4.5 Implementation and Evaluation

Implementation. We implemented a constraint solver MuVal that supports invariant synthesis and ranking function synthesis. For invariant synthesis, we apply an ordinary decision tree learning (see [25, 36, 44, 69, 127] for existing techniques). For ranking function synthesis, we implemented the algorithm in Section 4.4 with both eager and lazy strategies for halfspace selection. Our synthesizer uses well-founded relation explained in Section 4.4.5. Given a benchmark, we run our solver for both termination and non-termination verification in parallel, and when one of the two returns an answer, we stop the other and use the answer. MuVal is written in OCaml and uses Z3 as an SMT solver backend. We used clang and llvm2kittel [1] to convert C benchmarks to T2 [3] format files, which are then translated to pwCSP by MuVal. Our implementation is available at `https://github.com/hiroshi-unno/coar`.

Experiments. We evaluated our implementation MuVal on C benchmarks from Termination Competition 2020 (C Integer) [4]. We compared our tool with AProVE [22, 35], iRankFinder [18], and Ultimate Automizer [48]. Experiments are conducted on StarExec [2] (CentOS 7.7 (1908) on Intel(R)

Table 4.1: Numbers of solved benchmarks

	Yes	No	TO	U
MuVal (eager)	204	89	42	0
MuVal (lazy)	200	84	51	0
AProVE	216	100	16	3
iRankFinder	208	92[1]	0	34
Ultimate Automizer	180	83	2	70

Xeon(R) CPU E5-2609 0 @ 2.40GHz (2393 MHZ) with 263932744 kB main memory). The time limit was 300 seconds.

Results. Results are shown in Table 4.1. Yes/No/TO/U means the number of benchmarks that these tools could verify termination/could verify non-termination/could not answer within 300 seconds and timed out (TimeOut)/gave up before 300 seconds (Unknown), respectively. We also show scatter plots of runtime in Fig. 4.6.

MuVal was able to solve more benchmarks than Ultimate Automizer. Compared to iRankFinder, MuVal solved slightly fewer benchmarks, but was faster in a large number of benchmarks: 265 benchmarks were solved faster by MuVal, 68 by iRankFinder, and 2 were not solved by both tools within 300 seconds (here, we regard U (unknown) as 300 seconds). Compared to AProVE, MuVal solved fewer benchmarks. However, there are several benchmarks that MuVal could solve but AProVE could not. Among them is "TelAviv-Amir-Minimum_true-termination.c", which does require piecewise affine ranking functions. MuVal found a ranking function $f(x, y) = \textbf{if } x - y \geq 0 \textbf{ then } y \textbf{ else } x$, while AProVE timed out.

We also observed that using CEGIS with transition examples itself showed its strengths even for benchmarks that do not require piecewise affine ranking functions. Notably, there are three benchmarks that MuVal could solve but the other tools could not; they are examples that do not require segmentations. Further analysis of these benchmarks indicates the following strengths of our framework: (1) the ability to handle nonlinear constraints (to some extent) thanks to the example-based synthesis and the recent development of SMT solvers; and (2) the ability to find a long lasso-shaped non-terminating trace assembled from multiple transition examples. Details are described below.

A benchmark containing a nonlinear operation. Fig. 4.5a shows one of the benchmarks that contains a nonlinear operation `y = y * y` but admits a linear ranking function. Although handling nonlinear operation is difficult in general, MuVal was able to verify termination. The reason can be understood as follows. (1) Our validator can find transition examples for this benchmark thanks to the recent development of SMT solvers. (2) Our synthesizer can work as usual because it is example-based.

A benchmark that requires a long lasso-shaped non-terminating trace to prove non-termination. Fig. 4.5b is a benchmark that is non-terminating.

[1] We removed one benchmarks from the result of iRankFinder because the answer was wrong.

```
int main() {
  int x = ?;
  int y = 2;
  int res = 1;
  if (x < 0 || y < 1) { }
  else {
    while (x > y) {
      y = y*y;
      res = 2*res;
    }
  }
}
```

(a) A benchmark containing a nonlinear operation.

```
int main() {
  int i = ?;
  int range = 20;
  while (0 <= i && i <= range) {
    if (!(0 == i && i == range)) {
      if (i == range) {
        i = 0;
        range = range-1;
      } else {
        i = i+1;
      }
    }
  }
}
```

(b) A benchmark that requires a long lasso to prove non-termination

Figure 4.5: Some of the benchmarks from Termination Competition 2020 (C Integer).

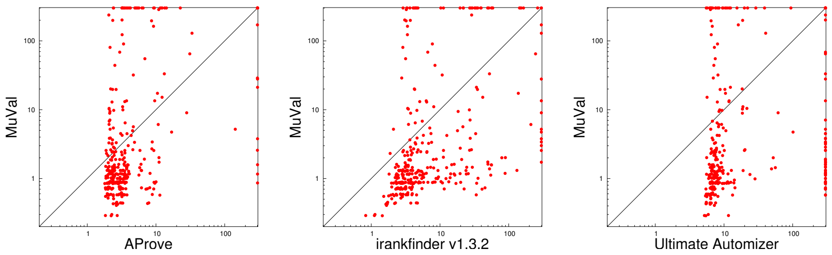


Figure 4.6: Scatter plots of runtime. Ultimate Automizer and AProVE sometimes gave up before the time limit, and such cases are regarded as 300s.

To prove non-termination of this benchmark, we need to find a long lasso-shaped trace that ends in the self loop when $\mathtt{i} = \mathtt{range} = 0$. Finding such a long lasso is difficult in general, but MuVal was able to find one in this benchmark: during CEGIS iterations, MuVal found the self loop (i.e. explicit cycle) and then collected transition examples that were needed to prove reachability of cycles.

Our method can naturally collect transition examples "backward" from the self loop (an explicit cycle). At the same time, our method collects transition examples "forward" from an initial state (in the benchmark of Fig. 4.5b, the pair of $\mathtt{i} = 10$ and $\mathtt{range} = 20$ is an initial state of the while loop). When transition examples that are collected backward and forward form a trace to the self loop, our method notice that the benchmark is non-terminating.

4.6 Related Work

There are a bunch of works that synthesize ranking functions via constraint solving. Among them is a counterexample-guided method like CEGIS [107]. CEGIS is sound but not guaranteed to be complete in general: even if a given constraint has a solution, CEGIS may fail to find the solution. A complete method for ranking function synthesis is proposed in [46]. They collect only extremal counterexamples instead of arbitrary transition examples to avoid in-finitely many examples. A limitation of their method is that the search space is limited to (lexicographic) affine ranking functions.

Another counterexample-guided method is proposed in [121] and implemented in SeaHorn. This method can synthesize piecewise affine functions, but their approach is quite different from ours. Given a program, they construct a *safety* property that the number of loop iterations does not exceed the value of a candidate ranking function. The safety property is checked by a verifier. If it is violated, then a trace is obtained as a counterexample and the candidate ranking function is updated by the counterexample. The main difference from our method is that their method uses trace examples while our method uses transition examples (which is less expensive to handle). FreqTerm [37] also uses the connection to safety property, but they exploit syntax-guided synthesis for synthesizing ranking functions.

Aside from counterexample-guided methods, constraint solving is widely studied for affine ranking functions [93], lexicographic affine ranking functions [12, 18, 78], and multiphase affine ranking functions [17, 19]. Their implementation includes RankFinder and iRankFinder. Farkas' lemma or Motzkin's transposition theorem are often used as a tool to transform $\exists\forall$-constraints to $\exists$-constraints. However, when we apply this technique to piecewise affine ranking functions, we get nonlinear constraints [78].

Abstract interpretation is also applied to segmented synthesis of ranking functions and implemented in FuncTion [120, 122, 123]. In this series of work, decision tree representation of ranking functions is used in [123] for better handling of disjunctions. Compared to their work, we believe that our method is more easily extensible to other theories than linear integer arithmetic as long as the theories are supported by SMT solvers (although such extensions are out of the scope of this thesis).

Other state-of-the-art termination verifiers include the following. Ultimate Automizer [48] is an automata-based method. It repeatedly finds a trace and computes a termination argument that contains the trace until termination arguments cover the set of all traces. Büchi automata are used to handle such traces. AProVE [22, 35] is based on term rewriting systems.

4.7 Conclusions and Future Work

In this chapter, we proposed a novel decision tree-based synthesizer for ranking functions, which is integrated into the CEGIS architecture. The key observation here was that we need to cope with explicit and implicit cycles contained in given examples. We designed a decision tree learning algorithm using the theoretical observation of the cycle detection theorem. We implemented the framework and observed that its performance is comparable to state-of-the-art termination analyzers. In particular, it solved three benchmarks that no other tool solved, a result that demonstrates the potential of the current combination of CEGIS, segmented synthesis, and transition examples.

We plan to extend our ranking function synthesizer to a synthesizer of piecewise affine ranking supermartingales. Ranking supermartingales [24] are probabilistic version of ranking functions and used for verification of almost-sure termination of probabilistic programs. Moreover, we would like to extend this further to ranking supermartingales for higher moments introduced in Chapter 5.

We also plan to implement a mechanism to automatically select a suitable set of halfspaces with which decision trees are built. In our ranking function synthesizer, intervals/octagons/octahedron/polyhedra can be used as the set of halfspaces. However, selecting an overly expressive set of halfspaces may cause the problem of overfitting [87] and result in poor performance. Therefore, applying heuristics that adjusts the expressiveness of halfspaces based on the current examples may improve the performance of our tool.

Chapter 5

Tail Probabilities of Randomized Programs via Higher Moments of Runtime

This chapter is about a method to give an upper bound of a tail probability of a randomized program. We provide a new notion, ranking supermartingales for higher moments based on an order-theoretic characterization of ranking supermartingales.

5.1 Preliminaries

We present some preliminary materials, including the definition of pCFGs (we use them as a model of randomized programs) and the definition of runtime.

Given topological spaces X and Y, let $\mathcal{B}(X)$ be the set of Borel sets on X and $\mathcal{B}(X, Y)$ be the set of Borel measurable functions $X \to Y$. We assume that the set $\mathbb{R}$ of reals, a finite set L and the set $[0, \infty]$ are equipped with the usual topology, the discrete topology, and the order topology, respectively. We use the induced Borel structures for these spaces. Given a measurable space X, let $\mathcal{D}(X)$ be the set of probability measures on X. For any $\mu \in \mathcal{D}(X)$, let $\mathrm{supp}(\mu)$ be the support of μ. We write $\mathbb{E}[X]$ for the expectation of a random variable X.

Our use of pCFGs follows recent works including [5].

Definition 5.1.1 (pCFG)**.** A *probabilistic control flow graph (pCFG)* is a tuple $\Gamma = (L, V, l_{\mathrm{init}}, \vec{x}_{\mathrm{init}}, \mapsto, \mathrm{Up}, \mathrm{Pr}, G)$ that consists of the following.

- A finite set L of *locations*. It is a disjoint union of sets L_D, L_P, L_n and L_A of *deterministic*, *probabilistic*, *nondeterministic* and *assignment* locations.

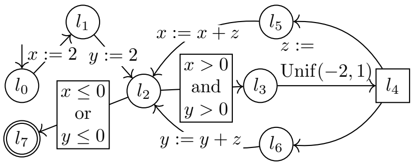
- A finite set V of *program variables*.

- An *initial location* $l_{\mathrm{init}} \in L$.

- An *initial valuation* $\vec{x}_{\text{init}} \in \mathbb{R}^V$.
- A *transition relation* $\mapsto \subseteq L \times L$ which is total (i.e. $\forall l.\, \exists l'.\, l \mapsto l'$).
- An *update function* $\text{Up} : L_A \to V \times \big(\mathcal{B}(\mathbb{R}^V, \mathbb{R}) \cup \mathcal{D}(\mathbb{R}) \cup \mathcal{B}(\mathbb{R})\big)$ for assignment.
- A family $\text{Pr} = (\text{Pr}_l)_{l \in L_P}$ of probability distributions, where $\text{Pr}_l \in \mathcal{D}(L)$, for probabilistic locations. We require that $l' \in \text{supp}(\text{Pr}_l)$ implies $l \mapsto l'$.
- A *guard function* $G : L_D \times L \to \mathcal{B}(\mathbb{R}^V)$ such that for each $l \in L_D$ and $\vec{x} \in \mathbb{R}^V$, there exists a unique location $l' \in L$ satisfying $l \mapsto l'$ and $\vec{x} \in G(l, l')$.

The update function can be decomposed into three functions $\text{Up}_D : L_{AD} \to V \times \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, $\text{Up}_P : L_{AP} \to V \times \mathcal{D}(\mathbb{R})$ and $\text{Up}_N : L_{AN} \to V \times \mathcal{B}(\mathbb{R})$, under a suitable decomposition $L_A = L_{AD} \cup L_{AP} \cup L_{AN}$ of assignment locations. The elements of L_{AD}, L_{AP} and L_{AN} represent *deterministic*, *probabilistic* and *nondeterministic* assignments, respectively. See e.g. [112].

An example of a pCFG is shown on the right. It models the program in Fig. 1.7. The node l_4 is a nondeterministic location. $\text{Unif}(-2, 1)$ is the uniform distribution on the interval $[-2, 1]$.



A *configuration* of a pCFG Γ is a pair $(l, \vec{x}) \in L \times \mathbb{R}^V$ of a location and a valuation. We regard the set $S = L \times \mathbb{R}^V$ of configurations is equipped with the product topology where L is equipped with the discrete topology. We say a configuration $(l', \vec{x}')$ is a *successor* of $(l, \vec{x})$, if $l \mapsto l'$ and the following hold.

- If $l \in L_D$, then $\vec{x}' = \vec{x}$ and $\vec{x} \in G(l, l')$.
- If $l \in L_N \cup L_P$, then $\vec{x}' = \vec{x}$.
- If $l \in L_A$, then $\vec{x}' = \vec{x}(x_j \leftarrow a)$, where $\vec{x}(x_j \leftarrow a)$ denotes the vector obtained by replacing the x_j-component of $\vec{x}$ by a. Here x_j is such that $\text{Up}(l) = (x_j, u)$, and a is chosen as follows: 1) $a = u(\vec{x})$ if $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$; 2) $a \in \text{supp}(u)$ if $u \in \mathcal{D}(\mathbb{R})$; and 3) $a \in u$ if $u \in \mathcal{B}(\mathbb{R})$.

An *invariant* of a pCFG Γ is a measurable set $I \in \mathcal{B}(S)$ such that $(l_{\text{init}}, \vec{x}_{\text{init}}) \in I$ and I is closed under taking successors (i.e. if $c \in I$ and c' is a successor of c then $c' \in I$). Use of invariants is a common technique in automated synthesis of supermartingales [5]: it restricts configuration spaces and thus makes the constraints on supermartingales weaker. It is also common to take an invariant as a measurable set [5]. A *run* of Γ is an infinite sequence of configurations $c_0 c_1 \ldots$ such that c_0 is the initial configuration $(l_{\text{init}}, \vec{x}_{\text{init}})$ and c_{i+1} is a successor of c_i for each i. Let $\text{Run}(\Gamma)$ be the set of runs of Γ.

A *scheduler* resolves nondeterminism: at a location in $L_N \cup L_{AN}$, it chooses a distribution of next configurations depending on the history of configurations visited so far. Given a pCFG Γ and a scheduler σ of Γ, a probability measure ν^{Γ}_{σ} on $\text{Run}(\Gamma)$ is defined in the usual manner. See Section B.1,B.2 for details.

Definition 5.1.2 (reaching time $T^{\Gamma}_{C}, T^{\Gamma}_{C,\sigma}$)**.** Let Γ be a pCFG and $C \subseteq S$ be a set of configurations called a *destination*. The *reaching time* to C is a function $T^{\Gamma}_{C} : \mathrm{Run}(\Gamma) \to [0, \infty]$ defined by $(T^{\Gamma}_{C})(c_0 c_1 \dots) = \mathrm{argmin}_{i \in \mathbb{N}}(c_i \in C)$. Fixing a scheduler σ makes T^{Γ}_{C} a random variable, since σ determines a probability measure ν^{Γ}_{σ} on $\mathrm{Run}(\Gamma)$. It is denoted by $T^{\Gamma}_{C,\sigma}$.

Runtimes of pCFGs are a special case of reaching times, namely to the set of terminating configurations.

The following higher moments are central to our framework. Recall that we are interested in demonic schedulers, i.e. those which make runtimes longer.

Definition 5.1.3 ($\mathbb{M}^{\Gamma,k}_{C,\sigma}$ and $\overline{\mathbb{M}}^{\Gamma,k}_{C}$)**.** Assume the setting of Def. 5.1.2, and let $k \in \mathbb{N}$ and $c \in S$. We write $\mathbb{M}^{\Gamma,k}_{C,\sigma}(c)$ for the k-th moment of the reaching time of Γ from c to C under the scheduler σ, i.e. that is, $\mathbb{M}^{\Gamma,k}_{C,\sigma}(c) = \mathbb{E}[(T^{\Gamma_c}_{C,\sigma})^k] = \int (T^{\Gamma_c}_{C})^k \, \mathrm{d}\nu^{\Gamma_c}_{\sigma}$ where Γ_c is a pCFG obtained from Γ by changing the initial configuration to c. Their supremum under varying σ is denoted by $\overline{\mathbb{M}}^{\Gamma,k}_{C} := \sup_{\sigma} \mathbb{M}^{\Gamma,k}_{C,\sigma}$.

5.2 Ranking Supermartingale for Higher Moments

We introduce one of the main contributions in the chapter, a notion of ranking supermartingale that overapproximates higher moments. It is motivated by the following observation: martingale-based reasoning about the second moment must concur with one about the first moment. We conduct a systematic theoretical extension that features an order-theoretic foundation and vector-valued supermartingales. The theory accommodates nondeterminism and continuous distributions, too. Detailed proofs are described in Section B.3.

The fully general theory for higher moments will be presented in Section 5.2.2; we present its restriction to the second moments in Section 5.2.1 for readability.

Prior to these, we review the existing theory of ranking supermartingales, through the lens of order-theoretic fixed points. In doing so we follow [112].

Definition 5.2.1 ("nexttime" operation $\overline{\mathbb{X}}$ (pre-expectation))**.** Given $\eta : S \to [0, \infty]$, let $\overline{\mathbb{X}}\eta : S \to [0, \infty]$ be the function defined as follows.

- If $l \in L_D$ and $\vec{x} \vDash G(l, l')$, then $(\overline{\mathbb{X}}\eta)(l, \vec{x}) = \eta(l', \vec{x})$.
- If $l \in L_P$, then $(\overline{\mathbb{X}}\eta)(l, \vec{x}) = \sum_{l \mapsto l'} \mathrm{Pr}_l(l')\eta(l', \vec{x})$.
- If $l \in L_N$, then $(\overline{\mathbb{X}}\eta)(l, \vec{x}) = \sup_{l \mapsto l'} \eta(l', \vec{x})$.
- If $l \in L_A$, $\mathrm{Up}(l) = (x_j, u)$ and $l \mapsto l'$,
  - if $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, then $(\overline{\mathbb{X}}\eta)(l, \vec{x}) = \eta(l', \vec{x}(x_j \leftarrow u(\vec{x})))$;
  - if $u \in \mathcal{D}(\mathbb{R})$, then $(\overline{\mathbb{X}}\eta)(l, \vec{x}) = \int_{\mathbb{R}} \eta(l', \vec{x}(x_j \leftarrow y)) \, \mathrm{d}u(y)$; and
  - if $u \in \mathcal{B}(\mathbb{R})$, then $(\overline{\mathbb{X}}\eta)(l, \vec{x}) = \sup_{y \in u} \eta(l', \vec{x}(x_j \leftarrow y))$.

Intuitively, $\overline{\mathbb{X}}\eta$ is the expectation of η after one transition. Nondeterminism is resolved by the maximal choice.

We define $F_1 : (S \to [0,\infty]) \to (S \to [0,\infty])$ as follows.

$$(F_1(\eta))(c) = \begin{cases} 1 + (\overline{\mathbb{X}}\eta)(c) & c \in I \setminus C \\ 0 & \text{otherwise} \end{cases} \quad \text{(Here "1+" accounts for time elapse)}$$

The function F_1 is an adaptation of the *Bellman operator*, a classic notion in the theory of Markov processes. A similar notion is used e.g. in [61]. The function space $(S \to [0,\infty])$ is a complete lattice structure, because $[0,\infty]$ is; moreover F_1 is easily seen to be monotone. It is not hard to see either that the expected reaching time $\overline{\mathbb{M}}_C^{\Gamma,1}$ to C coincides with the least fixed point μF_1.

The following theorem is fundamental in theoretical computer science.

Theorem 5.2.2 (Knaster–Tarski, [114])**.** Let $(L, \leq)$ be a complete lattice and $f : L \to L$ be a monotone function. The least fixed point μf is the least prefixed point, i.e. $\mu f = \min\{l \in L \mid f(l) \leq l\}$. □

The significance of the Knaster-Tarski theorem in verification lies in the induced proof rule: $f(l) \leq l \Rightarrow \mu f \leq l$. Instantiating to the expected reaching time $\overline{\mathbb{M}}_C^{\Gamma,1} = \mu F_1$, it means $F_1(\eta) \leq \eta \Rightarrow \overline{\mathbb{M}}_C^{\Gamma,1} \leq \eta$, i.e. an arbitrary prefixed point of F_1—which coincides with the notion of ranking supermartingale [23]—overapproximates the expected reaching time. This proves soundness of ranking supermartingales.

5.2.1 Ranking Supermartingales for the Second Moments

We extend ranking supermartingales to the second moments. It paves the way to a fully general theory (up to the K-th moments) in Section 5.2.2.

The key in the martingale-based reasoning of expected reaching times (i.e. first moments) was that they are characterized as the least fixed point of a function F_1. Here it is crucial that for an arbitrary random variable T, we have $\mathbb{E}[T+1] = \mathbb{E}[T]+1$ and therefore we can calculate $\mathbb{E}[T+1]$ from $\mathbb{E}[T]$. However, this is not the case for second moments. As $\mathbb{E}[(T+1)^2] = \mathbb{E}[T^2] + 2\mathbb{E}[T] + 1$, calculating the second moment requires not only $\mathbb{E}[T^2]$ but also $\mathbb{E}[T]$. This encourages us to define a vector-valued supermartingale.

Definition 5.2.3 (time-elapse function El_1)**.** A function $\mathrm{El}_1 : [0,\infty]^2 \to [0,\infty]^2$ is defined by $\mathrm{El}_1(x_1, x_2) = (x_1 + 1, x_2 + 2x_1 + 1)$.

Then, an extension of F_1 for second moments can be defined as a combination of the time-elapse function El_1 and the pre-expectation $\overline{\mathbb{X}}$.

Definition 5.2.4 (F_2)**.** Let I be an invariant and $C \subseteq I$ be a Borel set. We define $F_2 : (S \to [0,\infty]^2) \to (S \to [0,\infty]^2)$ by

$$(F_2(\eta))(c) = \begin{cases} (\overline{\mathbb{X}}(\mathrm{El}_1 \circ \eta))(c) & c \in I \setminus C \\ (0,0) & \text{otherwise.} \end{cases}$$

Here $\overline{\mathbb{X}}$ is applied componentwise: $(\overline{\mathbb{X}}(\eta_1, \eta_2))(c) = ((\overline{\mathbb{X}}\eta_1)(c), (\overline{\mathbb{X}}\eta_2)(c))$.

We can extend the complete lattice structure of $[0,\infty]$ to the function space $S \to [0,\infty]^2$ in a pointwise manner. It is a routine to prove that F_2 is monotone with respect to this complete lattice structure. Hence F_2 has the least fixed point. In fact, while $\overline{\mathbb{M}}_C^{\Gamma,1}$ was characterized as the least fixed point of F_1, a tuple $(\overline{\mathbb{M}}_C^{\Gamma,1}, \overline{\mathbb{M}}_C^{\Gamma,2})$ is *not* the least fixed point of F_2 (cf. Example 5.2.8 and Thm. 5.2.9). However, the least fixed point of F_2 *overapproximates* the tuple of moments.

Theorem 5.2.5. For any configuration $c \in I$, $(\mu F_2)(c) \geq (\overline{\mathbb{M}}_C^{\Gamma,1}(c), \overline{\mathbb{M}}_C^{\Gamma,2}(c))$. □

Let $T^{\Gamma}_{C,\sigma,n} = \min\{n, T^{\Gamma}_{C,\sigma}\}$. To prove the above theorem, we inductively prove

$$(F_2)^n(\bot)(c) \geq \left(\int T^{\Gamma_c}_{C,\sigma,n}\, \mathrm{d}\nu^{\Gamma_c}_{\sigma},\ \int (T^{\Gamma_c}_{C,\sigma,n})^2\, \mathrm{d}\nu^{\Gamma_c}_{\sigma}\right)$$

for each σ and n, and take the supremum. See Section B.3 for more details.

Like ranking supermartingale for first moments, ranking supermartingale for second moments is defined as a prefixed point of F_2, i.e. a function η such that $\eta \geq F_2(\eta)$. However, we modify the definition for the sake of implementation.

Definition 5.2.6 (ranking supermartingale for second moments)**.** A ranking supermartingale for second moments is a function $\eta : S \to \mathbb{R}^2$ such that: i) $\eta(c) \geq (\overline{\mathbb{X}}(\mathrm{El}_1 \circ \eta))(c)$ for each $c \in I \setminus C$; and ii) $\eta(c) \geq 0$ for each $c \in I$.

Here, the time-elapse function El_1 captures a positive decrease of the ranking supermartingale. Even though we only have inequality in Thm. 5.2.5, we can prove the following desired property of our supermartingale notion.

Theorem 5.2.7. If $\eta : S \to \mathbb{R}^2$ is a supermartingale for second moments, then $\left(\overline{\mathbb{M}}_C^{\Gamma,1}(c), \overline{\mathbb{M}}_C^{\Gamma,2}(c)\right) \leq \eta(c)$ for each $c \in I$. □

The following example and theorem show that we cannot replace $\geq$ with $=$ in Thm. 5.2.5 in general, but it is possible in the absence of nondeterminism.

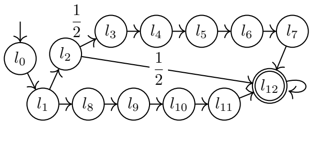
Example 5.2.8. The figure on the right shows a pCFG such that $l_2 \in L_P$ and all the other locations are in L_N, the initial location is l_0 and l_{12} is a terminating location. For the pCFG, the left-hand side of the inequality in Thm. 5.2.5 is $\mu F_2(l_0) = (6, 37.5)$. In contrast, if a scheduler σ takes a transition from l_1 to l_2 with probability p, $(\mathbb{M}^{\Gamma,1}_{C,\sigma}(l_0), \mathbb{M}^{\Gamma,2}_{C,\sigma}(l_0)) = \left(6 - \frac{1}{2}p, 36 - \frac{5}{2}p\right)$. Hence the right-hand side is $(\overline{\mathbb{M}}_C^{\Gamma,1}(l_0), \overline{\mathbb{M}}_C^{\Gamma,2}(l_0)) = (6, 36)$.

Theorem 5.2.9. If $L_N = L_{AN} = \emptyset$, $\forall c \in I.\, (\mu F_2)(c) = (\overline{\mathbb{M}}_C^{\Gamma,1}(c), \overline{\mathbb{M}}_C^{\Gamma,2}(c))$. □

5.2.2 Ranking Supermartingales for the Higher Moments

We extend the result in Section 5.2.1 to moments higher than second.

Firstly, the time-elapse function El_1 is generalized as follows.



Definition 5.2.10 (time-elapse function $\mathrm{El}_1^{K,k}$)**.** For $K \in \mathbb{N}$ and $k \in \{1, \ldots, K\}$, a function $\mathrm{El}_1^{K,k} : [0,\infty]^K \to [0,\infty]$ is defined by $\mathrm{El}_1^{K,k}(x_1, \ldots, x_K) = 1 + \sum_{j=1}^{k} \binom{k}{j} x_j$. Here $\binom{k}{j}$ is the binomial coefficient.

Again, a monotone function F_K is defined as a combination of the time-elapse function $\mathrm{El}_1^{K,k}$ and the pre-expectation $\overline{\mathbb{X}}$.

Definition 5.2.11 (F_K)**.** Let I be an invariant and $C \subseteq I$ be a Borel set. We define $F_K : (S \to [0,\infty]^K) \to (S \to [0,\infty]^K)$ by $F_K(\eta)(c) = (F_{K,1}(\eta)(c), \dots, F_{K,K}(\eta)(c))$, where $F_{K,k} : (S \to [0,\infty]^K) \to (S \to [0,\infty])$ is given by

$$(F_{K,k}(\eta))(c) = \begin{cases} (\overline{\mathbb{X}}(\mathrm{El}_1^{K,k} \circ \eta))(c) & c \in I \setminus C \\ 0 & \text{otherwise.} \end{cases}$$

As in Def. 5.2.6, we define a supermartingale as a prefixed point of F_K.

Definition 5.2.12 (ranking supermartingale for K-th moments)**.** We define $\eta_1, \dots, \eta_K : S \to \mathbb{R}$ by $(\eta_1(c), \dots, \eta_K(c)) = \eta(c)$. A *ranking supermartingale for K-th moments* is a function $\eta : S \to \mathbb{R}^K$ such that for each k, i) $\eta_k(c) \geq (\overline{\mathbb{X}}(\mathrm{El}_1^{K,k} \circ \eta_k))(c)$ for each $c \in I \setminus C$; and ii) $\eta_k(c) \geq 0$ for each $c \in I$.

For higher moments, we can prove an analogous result to Thm. 5.2.7.

Theorem 5.2.13. If η is a supermartingale for K-th moments, then for each $c \in I$, $(\overline{\mathbb{M}}_C^{\Gamma,1}(c), \dots, \overline{\mathbb{M}}_C^{\Gamma,K}(c)) \leq \eta(c)$. □

5.3 From Moments to Tail Probabilities

We discuss how to obtain upper bounds of tail probabilities of runtimes from upper bounds of higher moments of runtimes. Combined with the result in Section 5.2, it induces a martingale-based method for overapproximating tail probabilities.

We use a concentration inequality. There are many choices of concentration inequalities (see e.g. [21]), and we use a variant of Markov's inequality. We prove that the concentration inequality is not only sound but also complete in a sense.

Formally, our goal is to calculate is an upper bound of $\Pr(T_{C,\sigma}^{\Gamma} \geq d)$ for a given deadline $d > 0$, under the assumption that we know upper bounds $u_1, \dots, u_K$ of moments $\mathbb{E}[T_{C,\sigma}^{\Gamma}], \dots, \mathbb{E}[(T_{C,\sigma}^{\Gamma})^K]$. In other words, we want to overapproximate $\sup_\mu \mu([d,\infty])$ where μ ranges over the set of probability measures on $[0,\infty]$ satisfying $\left(\int x \,\mathrm{d}\mu(x), \dots, \int x^K \,\mathrm{d}\mu(x)\right) \leq (u_1, \dots, u_K)$.

To answer this problem, we use a generalized form of Markov's inequality.

Proposition 5.3.1 (see e.g. [21, Section 2.1])**.** Let X be a real-valued random variable and ϕ be a nondecreasing and nonnegative function. For any $d \in \mathbb{R}$ with $\phi(d) > 0$,

$$\Pr(X \geq d) \leq \frac{\mathbb{E}[\phi(X)]}{\phi(d)}.$$

□

By letting $\phi(x) = x^k$ in Prop 5.3.1, we obtain the following inequality. It gives an upper bound of the tail probability that is "tight."

Proposition 5.3.2. Let X be a nonnegative random variable. Assume $\mathbb{E}[X^k] \leq u_k$ for each $k \in \{0, \dots, K\}$. Then, for any $d > 0$,

$$\Pr(X \geq d) \leq \min_{0 \leq k \leq K} \frac{u_k}{d^k}.$$

Moreover, this upper bound is tight: for any $d > 0$, there exists a probability measure such that the above equation holds.

Proof. The former part is immediate from Prop 5.3.1. For the latter part, consider $\mu = p\delta_d + (1-p)\delta_0$ where δ_x is the Dirac measure at x and p is the value of the right-hand side of (5.3.2). □

By combining Thm. 5.2.13 with Prop. 5.3.2, we obtain the following corollary. We can use it for overapproximating tail probabilities.

Corollary 5.3.3. Let $\eta : S \to \mathbb{R}^K$ be a ranking supermartingale for K-th moments. For each scheduler σ and a deadline $d > 0$,

$$\Pr(T^{\Gamma}_{C,\sigma} \geq d) \leq \min_{0 \leq k \leq K} \frac{\eta_k(l_{\text{init}}, \vec{x}_{\text{init}})}{d^k}.$$

Here $\eta_0, \ldots, \eta_K$ are defined by $\eta_0(c) = 1$ and $\eta(c) = (\eta_1(c), \ldots, \eta_K(c))$. □

Note that if $K = 1$, Cor. 5.3.3 is essentially the same as [26, Thm. 4]. Note also that for each K there exists $d > 0$ such that

$$\frac{\eta_K(l_{\text{init}}, \vec{x}_{\text{init}})}{d^K} = \min_{0 \leq k \leq K} \frac{\eta_k(l_{\text{init}}, \vec{x}_{\text{init}})}{d^k}.$$

Hence higher moments become useful in overapproximating tail probabilities as d gets large. Later in Section 5.5, we demonstrate this fact experimentally.

5.4 Template-Based Synthesis Algorithm

We discuss an automated synthesis algorithm that calculates an upper bound for the k-th moment of the runtime of a pCFG using a supermartingale in Def. 5.2.6 or Def. 5.2.12. They are based on existing template-based algorithms for synthesizing a ranking supermartingale for first moments in [23,27,28]. Input to the algorithm is a pCFG Γ, an invariant I, a set $C \subseteq I$ of configurations, and a natural number K. Output is an upper bound of K-th moment.

5.4.1 Linear Template-Based Algorithm

Synthesis of a ranking supermartingale via reduction to an LP problem is discussed in [23, 28]. We adapt this for our supermartingales.

We first define some notions.

Definition 5.4.1. Let $V = \{x_1, \ldots, x_n\}$ be a set of variables. A *linear expression* over V is a formula of a form $a_1 x_{i_1} + \cdots + a_n x_{i_n} + b$ where $a_1, \ldots, a_n, b \in \mathbb{R}$ and $x_{i_1}, \ldots, x_{i_n} \in V$. We write $\mathbb{R}_{\text{lin}}[V]$ for the set of linear expressions. A *linear inequality* over V is a formula of a form $\varphi \geq 0$ where φ is a linear expression. A *linear conjunctive predicate* is a conjunction $\varphi_1 \geq 0 \land \cdots \land \varphi_p \geq 0$ of linear constraints, and a *linear predicate* is a disjunction $(\varphi_{1,1} \geq 0 \land \cdots \land \varphi_{1,p_1} \geq 0) \lor \cdots \lor (\varphi_{q,1} \geq 0 \land \cdots \land \varphi_{q,p_q} \geq 0)$ of linear conjunctive predicates. We define their semantics in the standard manner.

For a pCFG $\Gamma = (L, V, l_{\text{init}}, \vec{x}_{\text{init}}, \mapsto, \text{Up}, \Pr, G)$, a *linear expression map* (resp. *linear predicate map*) over Γ is a function that assigns a linear expression (resp. linear predicate) to each location of Γ. The semantics of the former is a function assigning a real number to each configuration, i.e. it has a type $L \times \mathbb{R}^V \to \mathbb{R}$, and that of the latter is a set of configurations, i.e. a subset of $L \times \mathbb{R}^V$. They are defined in the natural manners.

In the rest of this section, we describe a linear template-based synthesis algorithm for a pCFG Γ an invariant I, a set $C \subseteq I$ of configurations, and a natural number K. We assume that the input satisfies the following conditions. Similar conditions were assumed in [23, 28].

Assumption 5.4.2.

- For any $l \in L_A$ such that $\mathrm{Up}(l) = (x_j, u)$,
  - if $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, then u is represented by a linear expression over V;
  - if $u \in \mathcal{D}(\mathbb{R})$, the expectation of u is known; and
  - if $u \in \mathcal{B}(\mathbb{R})$, then u is represented by a linear predicate ϕ over $\{x_j\}$.
- For any $l \in L_D$ and $l' \in L$, $G(l, l') = [\![p]\!]$ is represented by a linear predicate over V.
- the invariant I is represented by a linear predicate map over Γ.
- the set C of terminal configurations is represented by a linear conjunctive predicate map.

Let $V = \{x_1, \ldots, x_n\}$ be the set of variables appearing in Γ. We first fix a *linear template* to a supermartingale. It is a family of formulas indexed by $i \in \{1, \ldots, K\}$ and $l \in L$ that have the following form:

$$\eta_i(l, \vec{x}) \ = \ a^l_{1,i} x_1 + \cdots + a^l_{n,i} x_n + b^l_i \,.$$

Here $a^l_{1,i}, \ldots, a^l_{n,i}, b^l_i$ are newly added variables called *parameters*. We write U for the set of all parameters, i.e. $U := \{a^l_{1,i}, \ldots, a^l_{n,i}, b^l_i \mid i \in \{1, \ldots, K\}, l \in L\}$. Note that if we fix a valuation $U \to \mathbb{R}$ of parameters, then each $\eta_i(l, \vec{x})$ reduces to a linear expression over V, and therefore $\eta_i(\_, \vec{x})$ can be regarded as a linear expression map $L \times \mathbb{R}^V \to \mathbb{R}_{\mathrm{lin}}[V]$. Our goal is to find a valuation $U \to \mathbb{R}$ so that a K-tuple $\big(\eta_1(\_, \vec{x}), \ldots, \eta_K(\_, \vec{x})\big)$ of linear expression maps become a ranking supermartingale for K-th moment (Definition 5.2.12).

To this end, we reduce the axioms of ranking supermartingale for K-th moments in Definition 5.2.12 to conditions over the parameters. We shall omit the detail, but it is not so hard to see that as a result of the reduction we obtain a conjunction of formulas of the following form:

$$\forall \vec{x} \in \mathbb{R}^V.\ \varphi_1 \rhd_1 0 \wedge \cdots \varphi_m \rhd_m 0 \implies \psi \geq 0 \,.$$

Here $\rhd_i \in \{\geq, >\}$, each φ_i is a linear expression without parameters, and ψ is a linear formula over V whose coefficients are linear expressions over U.

We next relax the strict inequalities as follows:

$$\forall \vec{x} \in \mathbb{R}^V.\ \varphi_1 \geq 0 \wedge \cdots \varphi_m \geq 0 \implies \psi \geq 0 \,.$$

It is easy to see that (5.4.1) implies (5.4.1). The same relaxation is also done in [23, 28].

Using matrices, we can represent a formula (5.4.1) as follows:

$$\forall \vec{x} \in \mathbb{R}^V.\ A\vec{x} \le b \implies c^T x \le d\,.$$

Here A is a matrix and b is a vector all of whose components are real numbers, and c is a vector and d is a scalar all of whose components are linear expressions over U. In [23, 28], a formula (5.4.1) is reduced to the following formula:

$$\exists \vec{y} \in \mathbb{R}^m.\ \exists y' \in \mathbb{R}.\ d - c^T x = \vec{y}^T\big(b - A\vec{x}\big) + y'\,.$$

Here m is the dimension of b. It is easy to see that (5.4.1) implies (5.4.1). By comparing the coefficients on both sides, we can see that (5.4.1) is equivalent to

$$\exists \vec{y} \in \mathbb{R}^m.\ A^T\vec{y} = c \wedge b^T y \le d\,.$$

Note that the resulting (in)equalities are linear with respect to parameters in U and $\vec{y}$. Hence its feasibility can be efficiently checked using a linear programming (LP) solver.

Recall that our goal is to calculate an upper bound of K-th moment. Hence we naturally want to minimize the upper bound $\eta_K(l_{\mathrm{init}}, \vec{x}_{\mathrm{init}})$ calculated by a supermartingale (see Thm. 5.2.13). We can achieve this goal by setting $\eta_K(l_{\mathrm{init}}, \vec{x}_{\mathrm{init}})$, a linear expression over U, to the objective function of the linear programming problem and ask the LP solver to minimize it.

A natural question would be about the converse of the implication (5.4.1) $\Rightarrow$ (5.4.1). The following theorem answers the question to some extent.

Theorem 5.4.3 (affine form of Farkas' lemma (see e.g. [103, Cor. 7.1h]))**.** Let $A \in \mathbb{R}^{n\times m}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. If $\{x \mid Ax \le b\}$ is not empty, the following two conditions are equivalent.

- $\forall x \in \mathbb{R}^n, Ax \le b \implies c^T x \le d$
- $\exists y \in \mathbb{R}^m, A^T y = c \wedge b^T y \le d$ □

We note that if a pCFG Γ has no program variable ($V = \emptyset$) and all the transitions are probabilistic (that is, if Γ is a Markov chain), the above method gives the exact value of moments. It is because the LP problem has the following obvious optimal solution: $\eta_k(l) =$ (the k-th moment of runtimes from location l).

5.4.2 Polynomial Template-based Algorithm

We consider fixing a polynomial template for a supermartingale. The algorithm in this section is based on [27].

Definition 5.4.4. Let $V = \{x_1, \ldots, x_n\}$ be a set of variables. A *monomial* is a formula of a form $x_{i_1}^{d_1} \ldots x_{i_p}^{d_p}$. We call $d_1 + \cdots + d_p$ a *degree* of the monomial, and write $\mathcal{M}_{\le d}$ for the set of monomials whose degrees are no greater than d. A *polynomial expression* (or simply a *polynomial*) over V is a formula of a form $a_1 m_1 + \cdots + a_q m_q + b$ where $a_1, \ldots, a_q, b \in \mathbb{R}$ and $m_1, \ldots, m_q$ are monomials. We write $\mathbb{R}[V]$ for the set of polynomial expressions over V. The notions of *polynomial inequality*, *polynomial conjunctive predicate*, *polynomial predicate*, *polynomial expression map* and *polynomial predicate map* are defined in a similar manner to Def. 5.4.1.

In the polynomial case, we assume that a pCFG Γ, an invariant I, a set $C \subseteq I$ of configurations and a natural number K satisfy the following conditions.

Assumption 5.4.5.

- For any $l \in L_A$, $\mathrm{Up}(l) = (x_j, u)$,
  - if $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, then u is represented by a polynomial expression over V
  - if $u \in \mathcal{D}(\mathbb{R})$, the K-th moment of u is known.
  - if $u \in \mathcal{B}(\mathbb{R})$, then u is represented by a polynomial predicate ϕ over $\{x_j\}$.
- For any $l \in L_D$, l', $G(l, l') = [\![p]\!]$ is represented by a polynomial predicate p over V.
- the invariant I is represented by a polynomial predicate map over Γ.
- the set C of terminal configurations is represented by a polynomial conjunctive predicate map.

The polynomial template-based synthesis algorithm is similar to the linear template-based one. In the polynomial case, the user have to fix the maximum degree d of the polynomial template. The algorithm first fixes a d-degree polynomial template for a supermartingale. It is a family of formulas indexed by $i \in \{1, \ldots, K\}$ and $l \in L$ that have the following form:

$$\eta_i(l, \vec{x}) \;=\; \sum_{m \in \mathcal{M}_{\leq d}} a^l_{m,i} m + b^l_i \,.$$

Each $a^l_{m,i}$ and b^l_i are newly added variables called *parameters*, and we write U for the set of all parameters. Our goal is to find a valuation $U \to \mathbb{R}$ so that a K-tuple $\big(\eta_1(\_, \vec{x}), \ldots, \eta_K(\_, \vec{x})\big)$ of polynomial expression maps is a ranking supermartingale for K-th moment (Definition 5.2.12).

Similarly to the linear case, the algorithm collects conditions on the parameters. It results in a conjunction of formulas of the following form:

$$\forall \vec{x} \in \mathbb{R}^V.\ \varphi_1 \rhd_1 0 \wedge \cdots \varphi_m \rhd_m 0 \implies \psi \geq 0\,.$$

Here $\rhd_i \in \{\geq, >\}$, each φ_i is a polynomial expression without parameters, and ψ is a polynomial formula over V whose coefficients are *linear* expressions over U. Relaxing the strict inequalities, we obtain the following:

$$\forall \vec{x} \in \mathbb{R}^V.\ \varphi_1 \geq 0 \wedge \cdots \varphi_m \geq 0 \implies \psi \geq 0\,.$$

To reduce (5.4.2) to a form that is solvable by a numerical method, we can use the notion of *sum-of-square polynomials* [27]. A polynomial expression f is said to be *sum-of-square* (SOS) if there exist polynomial expressions $g_1, \ldots, g_l$ such that $f = g_1^2 + \cdots + g_l^2$.

Obviously, a sum-of-square polynomial is nonnegative. Therefore the following formula is a sufficient condition for (5.4.2):

$$\exists \big(h_w : \text{sum-of-square}\big)_{w \in \{0,1\}^m}.\ \psi = \sum_{w \in \{0,1\}^m} h_w \cdot \varphi_1^{w_1} \cdot \cdots \cdot \varphi_m^{w_m}\,.$$

Here w_i denotes the i-th component of $w \in \{0,1\}^m$.

One of the reasons that sum-of-square is convenient is that it is characterized using the notion of *positive semidefinite matrix*.

Proposition 5.4.6 (see e.g. [52]). A polynomial expression f over V is sum-of-square if and only if there exist a vector $\vec{y}$ whose components are monomials over V and a positive semidefinite matrix A such that $f = \vec{y}^T A \vec{y}$. □

By the proposition above, existence of a valuation $U \to \mathbb{R}$ of parameters and sum-of-square polynomials as in (5.4.2) can be reduced to a *semidefinite programming* (SDP) problem. Likewise the linear case, by setting a linear expression $\eta_K(l_{\mathrm{init}}, \vec{x}_{\mathrm{init}})$ to the objective function, we can minimize it.

In the linear case, completeness was partially ensured by Farkas' lemma. In the polynomial case, the role is played by the following theorem.

Theorem 5.4.7 (Schmüdgen's Positivstellensatz [102]). Let $g, g_1, \ldots, g_m$ be polynomial expression over a set of variable V. If $\{\vec{x} \in \mathbb{R}^V \mid \bigwedge_{i=1}^m g_i \geq 0\}$ is compact, then the following conditions are equivalent:

- $\forall x \in \mathbb{R}^V.\ g_1 \geq 0 \wedge \cdots \wedge g_m \geq 0 \implies g > 0.$
- there exists a family $\{h_w\}_{w \in \{0,1\}^m}$ of sum-of-square polynomial expressions such that $g = \sum_{w \in \{0,1\}^m} h_w \cdot g_1^{w_1} \cdot \cdots \cdot g_m^{w_m}$. □

5.5 Experiments

We implemented two programs in OCaml to synthesize a supermartingale based on a) a linear template and b) a polynomial template. The programs translate a given randomized program to a pCFG and output an LP or SDP problem as described in Section 5.4. An invariant I and a terminal configuration C for the input program are specified manually. See e.g. [62] for automatic synthesis of an invariant. For linear templates, we have used GLPK (v4.65) [45] as an LP solver. For polynomial templates, we have used SOSTOOLS (v3.03) [108] (a sums of squares optimization tool that internally uses an SDP solver) on Matlab (R2018b). We used SDPT3 (v4.0) [105] as an SDP solver. The experiments were carried out on a Surface Pro 4 with an Intel Core i5-6300U (2.40GHz) and 8GB RAM. We tested our implementation for the following two programs and their variants, which were also used in the literature [28, 61]. Their code is in Section B.4.

Coupon collector's problem. A probabilistic model of collecting coupons enclosed in cereal boxes. There exist n types of coupons, and one repeatedly buy cereal boxes until all the types of coupons are collected. We consider two cases: (1-1) $n = 2$ and (1-2) $n = 4$. We tested the linear template program for them.

Random walk. We used three variants of 1-dimensional random walks: (2-1) integer-valued one, (2-2) real-valued one with assignments from continuous distributions, (2-3) with adversarial nondeterminism; and two variants of 2-dimensional random walks (2-4) and (2-5) with assignments from continuous distributions and adversarial nondeterminism. We tested both the linear and the polynomial template programs for these examples.

```
x := 200000000;
while true do
  if prob(0.7) then
  z := Unif(0,1);
  x := x - z
  else
  z := Unif(0,1);
  x := x + z
  fi;
  refute (x < 0)
od
```

Figure 5.1: A variant of (2-2).

Experimental results We measured execution times needed for Step 1 in Fig. 1.8. The results are in Table 5.1. Execution times are less than 0.2 seconds for linear template programs and several minutes for polynomial template programs. Upper bounds of tail probabilities obtained from Prop. 5.3.2 are in Fig. 5.2.

We can see that our method is applicable even with nondeterministic branching ((2-3), (2-4) and (2-5)) or assignments from continuous distributions ((2-2), (2-4) and (2-5)). We can use a linear template for bounding higher moments as long as there exists a supermartingale for higher moments representable by linear expressions ((1-1), (1-2) and (2-3)). In contrast, for (2-1), (2-2) and (2-4), only a polynomial template program found a supermartingale for second moments.

It is expectable that the polynomial template program gives a better bound than the linear one because a polynomial template is more expressive than a linear one. However, it did not hold for some test cases, probably because of numerical errors of the SDP solver. For example, (2-1) has a supermartingale for third moments that can be checked by a hand calculation, but the SDP solver returned "infeasible" in the polynomial template program. It appears that our program fails when large numbers are involved (e.g. the third moments of (2-1), (2-2) and (2-3)). We have also tested a variant of (2-1) where the initial position is multiplied by 10000. Then the SDP solver returned "infeasible" in the polynomial template program while the linear template program returns a nontrivial bound. Hence it seems that numerical errors are likely to occur to the polynomial template program when large numbers are involved.

Fig. 5.2 shows that the bigger the deadline d is, the more useful higher moments become (cf. a remark just after Cor. 5.3.3). For example, in (1-2), an upper bound of $\Pr(T^{\Gamma}_{C,\sigma} \geq 100)$ calculated from the upper bound of the first moment is 0.680 while that of the fifth moment is 0.105.

To show the merit of our method compared with sampling-based methods, we calculated a tail probability bound for a variant of (2-2) (shown in Fig. 5.1 on p. 112)) with a deadline $d = 10^{11}$. Because of its very long expected runtime, a sampling-based method would not work for it. In contrast, the linear template-based program gave an upper bound $\Pr(T^{\Gamma}_{C,\sigma} \geq 10^{11}) \leq 5000000025/10^{11} \approx 0.05$ in almost the same execution time as (2-2) (< 0.02 seconds).

Table 5.1: Upper bounds of the moments of runtimes. “-” indicates that the LP or SDP solver returned “infeasible”. The “degree” column shows the degree of the polynomial template used in the experiments.

		a) linear template		b) polynominal template		
	moment	upper bound	time (sec)	upper bound	time (sec)	degree
(1-1)	1st	13	0.012			
	2nd	201	0.019			
	3rd	3829	0.023			
(1-2)	1st	68	0.024			
	2nd	3124	0.054			
	3rd	171932	0.089			
	4th	12049876	0.126			
	5th	1048131068	0.191			
(2-1)	1st	20	0.024	20.0	24.980	2
	2nd	-	0.013	2320.0	37.609	2
	3rd	-	0.017	-	30.932	3
(2-2)	1st	75	0.009	75.0	33.372	2
	2nd	-	0.014	8375.0	73.514	2
	3rd	-	0.021	-	170.416	3
(2-3)	1st	62	0.020	62.0	40.746	2
	2nd	28605.4	0.038	6710.0	97.156	2
	3rd	19567043.36	0.057	-	35.427	3
(2-4)	1st	96	0.020	95.95	157.748	2
	2nd	-	0.029	10944.0	361.957	2
(2-5)	1st	90	0.022	-	143.055	2
	2nd	-	0.042	-	327.202	2

5.6 Related Work

Martingale-Based Analysis of Randomized Programs Martingale-based methods are widely studied for the termination analysis of randomized programs. One of the first is *ranking supermartingales*, introduced in [23] for proving almost sure termination. The theory of ranking supermartingales has since been extended actively: accommodating nondeterminism [5, 27, 28, 38], syntax-oriented composition of supermartingales [38], proving properties beyond termination/reachability [54], and so on. Automated template-based synthesis of supermartingales by constraint solving has been pursued, too [5, 23, 27, 28].

Other martingale-based methods that are fundamentally different from ranking supermartingales have been devised, too. They include: different notions of *repulsing supermartingales* for refuting termination (in [29, 112]; also studied in control theory [109]); and *multiply-scaled submartingales* for underapproximating reachability probabilities [112, 119]. See [112] for an overview.

In the literature on martingale-based methods, the one closest to this work is [26]. Among its contribution is the analysis of tail probabilities. It is done by either of the following combinations: 1) *difference-bounded* ranking supermartingales and the corresponding Azuma’s concentration inequality; and 2) (not necessarily difference-bounded) ranking supermartingales and Markov’s concentration inequality. When we compare these two methods with ours, the first method requires repeated martingale synthesis for different parameter values, which can pose a performance challenge. The second method corresponds

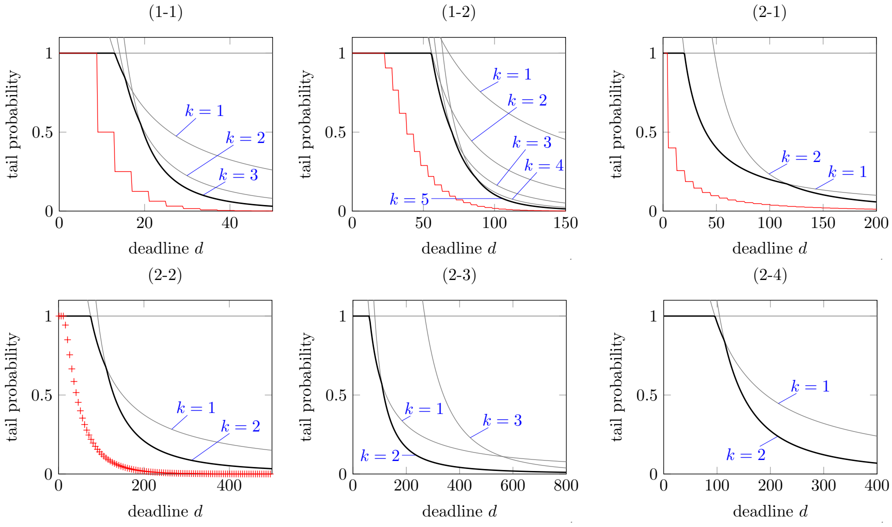


Figure 5.2: Upper bounds of the tail probabilities (except (2-5)). Each gray line is the value of $\frac{u_k}{d^k}$ where u_k is the best upper bound in Table 5.1 of k-th moments and d is a deadline. Each black line is the minimum of gray lines, i.e. the upper bound by Prop. 5.3.2. The red lines in (1-1), (1-2) and (2-1) show the true tail probabilities calculated analytically. The red points in (2-2) show tail probabilities calculated by Monte Carlo sampling where the number of trials is 100000000. We did not calculate the true tail probabilities nor approximate them for (2-4) and (2-5) because these examples seem difficult to do so due to nondeterminism.

to the restriction of our method to the first moment; recall that we showed the advantage of using higher moments, theoretically (Section 5.3) and experimentally (Section 5.5). See Section B.5.1 for detailed discussions. Implementation is lacking in [26], too.

We use Markov's inequality to calculate an upper bound of $\Pr(T_{\mathrm{run}} \geq d)$ from a ranking supermartingale. In [28], Hoeffding's and Bernstein's inequalities are used for the same purpose. As the upper bounds obtained by these inequalities are exponentially decreasing with respect to d, they are asymptotically tighter than our bound obtained by Markov's inequality, assuming that we use the same ranking supermartingale. However, Hoeffding's and Bernstein's inequalities are applicable to limited classes of ranking supermartingales (so-called difference-bounded and incremental ones, respectively). There exists a randomized program whose tail probability for runtimes is decreasing only polynomially (not exponentially, see Section B.6; this witnesses that there are cases where the methods in [28] do not apply but ours can.

The work [5] is also close to ours in that their supermartingales are vector-valued. The difference is in the orders: in [5] they use the *lexicographic* order between vectors, and they aim to prove almost sure termination. In contrast, we use the *pointwise* order between vectors, for overapproximating higher moments.

The Predicate-Transformer Approach to Runtime Analysis In the runtime/termination analysis of randomized programs, another principal line of work uses *predicate transformers* [15, 59, 61], following the precedent works on probabilistic predicate transformers such as [68, 85]. In fact, from the mathematical point of view, the main construct for witnessing runtime/termination in those predicate transformer calculi (called *invariants*, see e.g. in [61]) is essentially the same thing as ranking supermartingales. Therefore the difference between the martingale-based and predicate-transformer approaches is mostly the matter of presentation—the predicate-transformer approach is more closely tied to program syntax and has a stronger deductive flavor. It also seems that there is less work on automated synthesis in the predicate-transformer approach.

In the predicate-transformer approach, the work [59] is the closest to ours, in that it studies *variance* of runtimes of randomized programs. The main differences are as follows: 1) computing tail probabilities is not pursued [59]; 2) their extension from expected runtimes to variance involves an additional variable τ, which poses a challenge in automated synthesis as well as in generalization to even higher moments; and 3) they do not pursue automated analysis. See Section B.5.2 for further details.

Higher Moments of Runtimes Computing and using higher moments of runtimes of probabilistic systems—generalizing randomized programs—has been pursued before. In [30], computing moments of runtimes of *finite-state* Markov chains is reduced to a certain linear equation. In the study of randomized algorithms, the survey [33] collects a number of methods, among which are some tail probability bounds using higher moments. Unlike ours, none of these methods are language-based static ones. They do not allow automated analysis.

Other Potential Approaches to Tail Probabilities We discuss potential approaches to estimating tail probabilities, other than the martingale-based one.

Sampling is widely employed for approximating behaviors of probabilistic systems; especially so in the field of probabilistic programming languages, since exact symbolic reasoning is hard in presence of conditioning. See e.g. [115]. We also used sampling to estimate tail probabilities in (2-2), Fig. 5.2. The main advantages of our current approach over sampling are threefold: 1) our upper bounds come with a mathematical guarantee, while the sampling bounds can always be erroneous; 2) it requires ingenuity to sample programs with nondeterminism; and 3) programs whose execution can take millions of years can still be analyzed by our method in a reasonable time, without executing them. The latter advantage is shared by static, language-based analysis methods in general; see e.g. [15].

Another potential method is probabilistic model checkers such as PRISM [73]. Their algorithms are usually only applicable to finite-state models, and thus not to randomized programs in general. Nevertheless, fixing a deadline d can make the reachable part $S_{\leq d}$ of the configuration space S finite, opening up the possibility of use of model checkers. It is an open question how to do so precisely, and the following challenges are foreseen: 1) if the program contains continuous distributions, the reachable part $S_{\leq d}$ becomes infinite; 2) even if $S_{\leq d}$ is finite, one has to repeat (supposedly expensive) runs of a model checker for each choice of d. In contrast, in our method, an upper bound for the tail probability $\Pr(T_{\mathrm{run}} \geq d)$ is symbolically expressed as a function of d (Prop. 5.3.2). Therefore, estimating tail probabilities for varying d is computationally cheap.

5.7 Conclusions and Future Work

We provided a technique to obtain an upper bound of the tail probability of runtimes given a randomized algorithm and a deadline. We first extended the ordinary ranking supermartingale notion using the order-theoretic characterization so that it can calculate upper bounds of higher moments of runtimes for randomized programs. Then by using a suitable concentration inequality, we introduced a method to calculate an upper bound of tail probabilities from upper bounds of higher moments. Our method is not only sound but also complete in a sense. Our method was obtained by combining our supermartingale and the concentration inequality. We also implemented an automated synthesis algorithm and demonstrated the applicability of our framework.

Future Work Example 5.2.8 shows that our supermartingale is not complete: it sometimes fails to give a tight bound for higher moments. Studying and improving the incompleteness is one possible direction of future work. For example, the following questions would be interesting: Can bounds given by our supermartingale be arbitrarily bad? Can we remedy the completeness by restricting the type of nondeterminism? Can we define a complete supermartingale?

Making our current method compositional is another direction of future research. Use of continuations, as in [60], can be a technical solution.

We are also interested in improving the implementation. The polynomial template program failed to give an upper bound for higher moments because of numerical errors (see Section 5.5). We wish to remedy this situation. There exist several studies for using numerical solvers for verification without affected by numerical errors [55–57, 95, 96]. We might make use of these works for improvements.

Appendix A

Full Definition of the Underlying Type System and the Dependent Refinement Type System

A.1 Underlying Type System

Our underlying type system in 2.4 is based on [9] but with minor modifications. In this section, we list all the typing rules and describe the interpretation of types and terms.

A.1.1 Typing Rules

Well-Formed Contexts

$$\frac{}{\vdash \diamond} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A \qquad x \notin \mathrm{Vars}(\Gamma)}{\vdash \Gamma, x : A}$$

Definitional Equality for Contexts

$$\frac{}{\vdash \diamond = \diamond} \qquad \frac{\vdash \Gamma_1 = \Gamma_2 \qquad \Gamma_1 \vdash A = B \qquad \Gamma_2 \vdash B \qquad x \notin \mathrm{Vars}(\Gamma_1) \cup \mathrm{Vars}(\Gamma_2)}{\vdash \Gamma_1, x : A = \Gamma_2, x : B}$$

Definitional Equality

Reflexivity.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A = A} \qquad \frac{\Gamma \vdash \underline{C}}{\Gamma \vdash \underline{C} = \underline{C}} \qquad \frac{\Gamma \vdash V \ : \ A}{\Gamma \vdash V = V \ : \ A} \qquad \frac{\Gamma \vdash M \ : \ \underline{C}}{\Gamma \vdash M = M \ : \ \underline{C}}$$

Symmetry.

$$\frac{\Gamma \vdash B = A}{\Gamma \vdash A = B} \qquad \frac{\Gamma \vdash \underline{D} = \underline{C}}{\Gamma \vdash \underline{C} = \underline{D}} \qquad \frac{\Gamma \vdash W = V \ : \ A}{\Gamma \vdash V = W \ : \ A} \qquad \frac{\Gamma \vdash N = M \ : \ \underline{C}}{\Gamma \vdash M = N \ : \ \underline{C}}$$

Transitivity.

$$\frac{\Gamma \vdash V_1 = V_2 \ : \ A \qquad \Gamma \vdash V_2 = V_3 \ : \ A}{\Gamma \vdash V_1 = V_3 \ : \ A}$$

$$\frac{\Gamma \vdash M_1 = M_2 \ : \ \underline{C} \qquad \Gamma \vdash M_2 = M_3 \ : \ \underline{C}}{\Gamma \vdash M_1 = M_3 \ : \ \underline{C}} \qquad \frac{\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash A_2 = A_3}{\Gamma \vdash A_1 = A_3}$$

$$\frac{\Gamma \vdash \underline{C}_1 = \underline{C}_2 \qquad \Gamma \vdash \underline{C}_2 = \underline{C}_3}{\Gamma \vdash \underline{C}_1 = \underline{C}_3}$$

Conversion

$$\frac{\Gamma_2 \vdash A \qquad \vdash \Gamma_1 = \Gamma_2}{\Gamma_1 \vdash A} \qquad \frac{\Gamma_2 \vdash \underline{C} \qquad \vdash \Gamma_1 = \Gamma_2}{\Gamma_1 \vdash \underline{C}} \qquad \frac{\Gamma_2 \vdash A = B \qquad \vdash \Gamma_1 = \Gamma_2}{\Gamma_1 \vdash A = B}$$

$$\frac{\Gamma_2 \vdash \underline{C} = \underline{D} \qquad \vdash \Gamma_1 = \Gamma_2}{\Gamma_1 \vdash \underline{C} = \underline{D}} \qquad \frac{\vdash \Gamma_1 = \Gamma_2 \qquad \Gamma_2 \vdash V \ : \ A \qquad \Gamma_1 \vdash A = B}{\Gamma_1 \vdash V \ : \ B}$$

$$\frac{\vdash \Gamma_1 = \Gamma_2 \qquad \Gamma_2 \vdash M \ : \ \underline{C} \qquad \Gamma_1 \vdash \underline{C} = \underline{D}}{\Gamma_1 \vdash M \ : \ \underline{D}}$$

Variables

$$\frac{\vdash \Gamma_1, x : A, \Gamma_2}{\Gamma_1, x : A, \Gamma_2 \vdash x \ : \ A}$$

Value constants

$$\frac{\vdash \Gamma \qquad \diamond \vdash \mathrm{ty}(c)}{\Gamma \vdash c_{\mathrm{ty}(c)} \ : \ \mathrm{ty}(c)}$$

Unit Type

$$\frac{\vdash \Gamma}{\Gamma \vdash 1} \qquad \frac{\vdash \Gamma}{\Gamma \vdash * \ : \ 1} \qquad \frac{\Gamma \vdash V \ : \ 1}{\Gamma \vdash V = * \ : \ 1}$$

Base Types

$$\frac{\Gamma \vdash V \ : \ A \qquad b : A \to \mathrm{Type} \qquad \diamond \vdash A}{\Gamma \vdash b_A(V)}$$

$$\frac{\Gamma \vdash V = W \ : \ A \qquad b : A \to \mathrm{Type} \qquad \diamond \vdash A}{\Gamma \vdash b_A(V) = b_A(W)}$$

Value Σ-Types

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Sigma x{:}A.B} \qquad \frac{\Gamma \vdash V \ : \ A \qquad \Gamma, x : A \vdash B \qquad \Gamma \vdash W \ : \ B[V/x]}{\Gamma \vdash \langle V, W \rangle \ : \ \Sigma x{:}A.B}$$

$$\frac{\Gamma \vdash V \ : \ \Sigma x{:}A.B \qquad \Gamma, z : \Sigma x{:}A.B \vdash \underline{C} \qquad \Gamma, x : A, y : B \vdash M \ : \ \underline{C}[\langle x, y \rangle/z]}{\Gamma \vdash \mathbf{pm}\ V\ \mathbf{as}\ \langle x : A, y : B \rangle\ \mathbf{in}_{z.\underline{C}}\ M \ : \ \underline{C}[V/z]}$$

$$\frac{\Gamma \vdash A_1 = A_2 \qquad \Gamma, x : A_1 \vdash B_1 = B_2}{\Gamma \vdash \Sigma x : A_1.B_1 = \Sigma x : A_2.B_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash V_1 = V_2 \ : \ A_2 \\ \Gamma, x : A_1 \vdash B_1 = B_2 \qquad \Gamma \vdash W_1 = W_2 \ : \ B_2[V_2/x]\end{array}}{\Gamma \vdash \langle V_1, W_1 \rangle = \langle V_2, W_2 \rangle \ : \ \Sigma x : A_2.B_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 = A_2 \qquad \Gamma, x : A_1 \vdash B_1 = B_2 \qquad \Gamma, z : \Sigma x{:}A_1.B_1 \vdash \underline{C}_1 = \underline{C}_2 \\ \Gamma \vdash V_1 = V_2 \ : \ \Sigma x{:}A_2.B_2 \\ \Gamma, x : A_1, y : B_1 \vdash M_1 = M_2 \ : \ \underline{C}_2[\langle x, y \rangle_{(x:A_2).B_2}/z]\end{array}}{\begin{array}{l}\Gamma \vdash \mathbf{pm}\ V_1\ \mathbf{as}\ \langle x : A_1, y : B_1 \rangle\ \mathbf{in}_{z.\underline{C}_1}\ M_1 \\ \qquad = \mathbf{pm}\ V_2\ \mathbf{as}\ \langle x : A_2, y : B_2 \rangle\ \mathbf{in}_{z.\underline{C}_2}\ M_2 \ : \ \underline{C}_2[V_2/z]\end{array}}$$

$$\frac{\begin{array}{c}\Gamma, z : \Sigma x{:}A.B \vdash \underline{C} \\ \Gamma \vdash V \ : \ A \qquad \Gamma \vdash W \ : \ B[V/x] \qquad \Gamma, x : A, y : B \vdash M \ : \ \underline{C}[\langle x, y \rangle/z]\end{array}}{\Gamma \vdash \mathbf{pm}\ \langle V, W \rangle\ \mathbf{as}\ \langle x : A, y : B \rangle\ \mathbf{in}_{z.\underline{C}}\ M = M[V/x][W/y] \ : \ \underline{C}[\langle V, W \rangle/z]}$$

$$\frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x : A \vdash B \\ \Gamma \vdash V \ : \ \Sigma x{:}A.B \qquad \Gamma, z : \Sigma x{:}A.B \vdash \underline{C} \qquad \Gamma, z : \Sigma x{:}A.B \vdash M \ : \ \underline{C}\end{array}}{\Gamma \vdash \mathbf{pm}\ V\ \mathbf{as}\ \langle x : A, y : B \rangle\ \mathbf{in}_{z.\underline{C}}\ M[\langle x, y \rangle/z] = M[V/z] \ : \ \underline{C}[V/z]}$$

Thunked Computation

$$\frac{\Gamma \vdash \underline{C}}{\Gamma \vdash U\underline{C}} \qquad \frac{\Gamma \vdash M \ : \ \underline{C}}{\Gamma \vdash \mathbf{thunk}\ M \ : \ U\underline{C}} \qquad \frac{\Gamma \vdash V \ : \ U\underline{C}}{\Gamma \vdash \mathbf{force}_{\underline{C}}\ V \ : \ \underline{C}} \qquad \frac{\Gamma \vdash \underline{C}_1 = \underline{C}_2}{\Gamma \vdash U\underline{C}_1 = U\underline{C}_2}$$

$$\frac{\Gamma \vdash M_1 = M_2 \ : \ \underline{C}}{\Gamma \vdash \mathbf{thunk}\ M_1 = \mathbf{thunk}\ M_2 \ : \ U\underline{C}} \qquad \frac{\Gamma \vdash \underline{C}_1 = \underline{C}_2 \qquad \Gamma \vdash V_1 = V_2 \ : \ U\underline{C}_2}{\Gamma \vdash \mathbf{force}_{\underline{C}_1}\ V_1 = \mathbf{force}_{\underline{C}_2}\ V_2 \ : \ \underline{C}_2}$$

$$\frac{\Gamma \vdash V \ : \ U\underline{C}}{\Gamma \vdash \mathbf{thunk}\ (\mathbf{force}_{\underline{C}}\ V) = V \ : \ U\underline{C}} \qquad \frac{\Gamma \vdash M \ : \ \underline{C}}{\Gamma \vdash \mathbf{force}_{\underline{C}}\ (\mathbf{thunk}\ M) = M \ : \ \underline{C}}$$

Return

$$\frac{\Gamma \vdash A}{\Gamma \vdash FA} \qquad \frac{\Gamma \vdash V \ : \ A}{\Gamma \vdash \mathbf{return}\ V \ : \ FA}$$

$$\frac{\Gamma \vdash M \ : \ FA \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x : A \vdash N \ : \ \underline{C}}{\Gamma \vdash M\ \mathbf{to}\ x : A\ \mathbf{in}_{\underline{C}}\ N \ : \ \underline{C}}$$

$$\frac{\Gamma \vdash A_1 = A_2}{\Gamma \vdash FA_1 = FA_2} \qquad \frac{\Gamma \vdash V_1 = V_2 \ : \ A}{\Gamma \vdash \mathbf{return}\ V_1 = \mathbf{return}\ V_2 \ : \ FA}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash M_1 = M_2 \ : \ FA_2 \\ \Gamma \vdash \underline{C}_1 = \underline{C}_2 \qquad \Gamma, x : A_1 \vdash N_1 = N_2 \ : \ \underline{C}_2\end{array}}{\Gamma \vdash M_1\ \mathbf{to}\ x : A_1\ \mathbf{in}_{\underline{C}_1}\ N_1 = M_2\ \mathbf{to}\ x : A_2\ \mathbf{in}_{\underline{C}_2}\ N_2 \ : \ \underline{C_2}}$$

$$\frac{\Gamma \vdash V \ : \ A \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x : A \vdash M \ : \ \underline{C}}{\Gamma \vdash \mathbf{return}\ V\ \mathbf{to}\ x : A\ \mathbf{in}_{\underline{C}}\ M = M[V/x] \ : \ \underline{C}[V/x]}$$

$$\frac{\Gamma \vdash M \ : \ FA}{\Gamma \vdash M\ \mathbf{to}\ x : A\ \mathbf{in}_{\underline{C}}\ \mathbf{return}\ x = M \ : \ FA}$$

$$\frac{\begin{array}{c}\Gamma \vdash M_1 \ : \ FA_1 \\ \Gamma \vdash A_2 \qquad \Gamma, x : A_1 \vdash M_2 \ : \ FA_2 \qquad \Gamma \vdash \underline{C} \qquad \Gamma, y : A_2 \vdash M_3 \ : \ \underline{C}\end{array}}{\begin{array}{l}\Gamma \vdash (M_1\ \mathbf{to}\ x : A_1\ \mathbf{in}_{FA_2}\ M_2)\ \mathbf{to}\ y : A_2\ \mathbf{in}_{\underline{C}}\ M_3 \\ \qquad = M_1\ \mathbf{to}\ x : A_1\ \mathbf{in}_{FA_2}\ (M_2\ \mathbf{to}\ y : A_2\ \mathbf{in}_{\underline{C}}\ M_3) \ : \ \underline{C}\end{array}}$$

Computational Π-Types

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash \underline{C}}{\Gamma \vdash \Pi x{:}A.\underline{C}} \qquad \frac{\Gamma, x : A \vdash M \ : \ \underline{C}}{\Gamma \vdash \lambda x : A.M \ : \ \Pi x{:}A.\underline{C}}$$

$$\frac{\Gamma, x : A \vdash \underline{C} \qquad \Gamma \vdash M \ : \ \Pi x{:}A.\underline{C} \qquad \Gamma \vdash V \ : \ A}{\Gamma \vdash M(V)_{(x:A).\underline{C}} \ : \ \underline{C}[V/x]}$$

$$\frac{\Gamma \vdash A_2 = A_1 \qquad \Gamma, x : A_2 \vdash \underline{C}_1 = \underline{C}_2}{\Gamma \vdash \Pi x{:}A_1.\underline{C}_1 = \Pi x{:}A_2.\underline{C}_2}$$

$$\frac{\Gamma \vdash A_1 = A_2 \qquad \Gamma, x : A_1 \vdash M_1 = M_2 \ : \ \underline{C}}{\Gamma \vdash \lambda x : A_1.M_1 = \lambda x : A_2.M_2 \ : \ \Pi x{:}A_1.\underline{C}}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_2 = A_1 \qquad \Gamma, x : A_2 \vdash \underline{C}_1 = \underline{C}_2 \\ \Gamma \vdash M_1 = M_2 \ : \ \Pi x{:}A_2.\underline{C}_2 \qquad \Gamma \vdash V_1 = V_2 \ : \ A_2\end{array}}{\Gamma \vdash M_1(V_1)_{(x:A_1).\underline{C}_1} = M_2(V_2)_{(x:A_2).\underline{C}_2} \ : \ \underline{C}_1[V_1/x]}$$

$$\frac{\Gamma, x : A \vdash M \ : \ \underline{C} \qquad \Gamma \vdash V \ : \ A}{\Gamma \vdash (\lambda x : A.M)(V)_{(x:A).\underline{C}} = M[V/x] \ : \ \underline{C}[V/x]}$$

$$\frac{\Gamma, x : A \vdash \underline{C} \qquad \Gamma \vdash M \ : \ \Pi x{:}A.\underline{C}}{\Gamma \vdash M = \lambda x : A.M(x)_{(x:A).\underline{C}} \ : \ \Pi x{:}A.\underline{C}}$$

Fibred Coproduct Types

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A + B} \qquad \frac{\Gamma \vdash V \ :\ A \qquad \Gamma \vdash B}{\Gamma \vdash \mathbf{inl}_{A+B}\ V \ :\ A + B} \qquad \frac{\Gamma \vdash V \ :\ B \qquad \Gamma \vdash A}{\Gamma \vdash \mathbf{inr}_{A+B}\ V \ :\ A + B}$$

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash \underline{C} \qquad \Gamma \vdash V \ :\ A + B \\ \Gamma, x : A \vdash M \ :\ \underline{C}[\mathbf{inl}_{A+B}\ x/z] \qquad \Gamma, y : B \vdash N \ :\ \underline{C}[\mathbf{inr}_{A+B}\ y/z]\end{array}}{\Gamma \vdash \mathbf{case}\ V\ \mathbf{of}_{z.\underline{C}}\ (\mathbf{inl}\ (x : A) \mapsto M, \mathbf{inr}\ (y : B) \mapsto N) \ :\ \underline{C}[V/z]}$$

$$\frac{\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash B_1 = B_2}{\Gamma \vdash A_1 + B_1 = A_2 + B_2}$$

$$\frac{\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash B_1 = B_2 \qquad \Gamma \vdash V_1 = V_2 \ :\ A_2}{\Gamma \vdash \mathbf{inl}_{A_1+B_1}\ V_1 = \mathbf{inl}_{A_2+B_2}\ V_2 \ :\ A_2 + B_2}$$

$$\frac{\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash B_1 = B_2 \qquad \Gamma \vdash V_1 = V_2 \ :\ B_2}{\Gamma \vdash \mathbf{inr}_{A_1+B_1}\ V_1 = \mathbf{inr}_{A_2+B_2}\ V_2 \ :\ A_2 + B_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 = A_2 \qquad \Gamma \vdash B_1 = B_2 \qquad \Gamma, z : A_1 + B_1 \vdash \underline{C}_1 = \underline{C}_2 \\ \Gamma \vdash V_1 = V_2 \ :\ A_1 + B_1 \qquad \Gamma, x : A_1 \vdash M_1 = M_2 \ :\ \underline{C}_2[\mathbf{inl}_{A_2+B_2}\ x/z] \\ \Gamma, y : B_1 \vdash N_1 = N_2 \ :\ \underline{C}_2[\mathbf{inr}_{A_2+B_2}\ y/z]\end{array}}{\begin{array}{l}\Gamma \vdash \mathbf{case}\ V_1\ \mathbf{of}_{z.\underline{C}_1}\ (\mathbf{inl}\ (x : A_1) \mapsto M_1, \mathbf{inr}\ (y : B_1) \mapsto N_1) \\ \qquad = \mathbf{case}\ V_2\ \mathbf{of}_{z.\underline{C}_2}\ (\mathbf{inl}\ (x : A_2) \mapsto M_2, \mathbf{inr}\ (y : B_2) \mapsto N_2) \ :\ \underline{C}_2[V_2/z]\end{array}}$$

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash \underline{C} \qquad \Gamma \vdash V \ :\ A \\ \Gamma, x : A \vdash M \ :\ \underline{C}[\mathbf{inl}_{A+B}\ x/z] \qquad \Gamma, y : B \vdash N \ :\ \underline{C}[\mathbf{inr}_{A+B}\ y/z]\end{array}}{\begin{array}{l}\Gamma \vdash \mathbf{case}\ (\mathbf{inl}_{A+B}\ V)\ \mathbf{of}_{z.\underline{C}}\ (\mathbf{inl}\ (x : A) \mapsto M, \mathbf{inr}\ (y : B) \mapsto N) \\ \qquad = M[V/x] \ :\ \underline{C}[\mathbf{inl}_{A+B}\ V/z]\end{array}}$$

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash \underline{C} \qquad \Gamma \vdash V \ :\ B \\ \Gamma, x : A \vdash M \ :\ \underline{C}[\mathbf{inl}_{A+B}\ x/z] \qquad \Gamma, y : B \vdash N \ :\ \underline{C}[\mathbf{inr}_{A+B}\ y/z]\end{array}}{\begin{array}{l}\Gamma \vdash \mathbf{case}\ (\mathbf{inr}_{A+B}\ V)\ \mathbf{of}_{z.\underline{C}}\ (\mathbf{inl}\ (x : A) \mapsto M, \mathbf{inr}\ (y : B) \mapsto N) \\ \qquad = N[V/y] \ :\ \underline{C}[\mathbf{inr}_{A+B}\ V/z]\end{array}}$$

$$\frac{\Gamma, z : A + B \vdash \underline{C} \qquad \Gamma \vdash V \ :\ A + B \qquad \Gamma, z : A + B \vdash M \ :\ \underline{C}}{\begin{array}{l}\Gamma \vdash \mathbf{case}\ V\ \mathbf{of}_{z.\underline{C}}\ (\mathbf{inl}\ (x : A) \mapsto M[\mathbf{inl}_{A+B}\ x/z], \mathbf{inr}\ (y : B) \mapsto M[\mathbf{inr}_{A+B}\ y/z]) \\ \qquad = M[V/z] \ :\ \underline{C}[V/z]\end{array}}$$

A.1.2 Semantics

Given a fibred adjunction model (Definition 2.4.1), we define the interpretation $[\![-]\!]$ of types and terms as follows (see also [9]). These definitions mean that if the upper part is defined, then the lower part is also defined.

Contexts

$$\frac{}{[\![\diamond]\!] = 1} \qquad \frac{[\![\Gamma; A]\!] \in \mathbb{E}_{[\![\Gamma]\!]} \qquad x \notin \mathrm{Vars}(\Gamma)}{[\![\Gamma, x : A]\!] = \{[\![\Gamma; A]\!]\}}$$

Types

$$\frac{b : A \to \mathrm{Type} \qquad \llbracket \diamond; A \rrbracket \in \mathbb{E}_1 \qquad \llbracket b \rrbracket \in \mathbb{E}_{\{\llbracket \diamond; A \rrbracket\}} \qquad \llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to !^*_{\llbracket \Gamma \rrbracket} \llbracket \diamond; A \rrbracket}{\llbracket \Gamma; b(V) \rrbracket = (s\llbracket \Gamma; V \rrbracket)^* \{\overline{!_{\llbracket \Gamma \rrbracket}}(\llbracket \diamond; A \rrbracket)\}^* \llbracket b \rrbracket}$$

$$\frac{\llbracket \Gamma \rrbracket \in \mathbb{B}}{\llbracket \Gamma; 1 \rrbracket = 1\llbracket \Gamma \rrbracket} \qquad \frac{\llbracket \Gamma; A \rrbracket \in \mathbb{E}_{\llbracket \Gamma \rrbracket} \qquad \llbracket \Gamma, x : A; B \rrbracket \in \mathbb{E}_{\llbracket \Gamma, x:A \rrbracket}}{\llbracket \Gamma; \Sigma x{:}A.B \rrbracket = \coprod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; B \rrbracket}$$

$$\frac{\llbracket \Gamma; \underline{C} \rrbracket}{\llbracket \Gamma; U\underline{C} \rrbracket = U\llbracket \Gamma; \underline{C} \rrbracket} \qquad \frac{\llbracket \Gamma; A \rrbracket}{\llbracket \Gamma; FA \rrbracket = F\llbracket \Gamma; A \rrbracket}$$

$$\frac{\llbracket \Gamma; A \rrbracket \qquad \llbracket \Gamma, x : A; \underline{C} \rrbracket}{\llbracket \Gamma; \Pi x{:}A.\underline{C} \rrbracket = \prod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; \underline{C} \rrbracket} \qquad \frac{\llbracket \Gamma; A \rrbracket \qquad \llbracket \Gamma; B \rrbracket}{\llbracket \Gamma; A + B \rrbracket = \llbracket \Gamma; A \rrbracket + \llbracket \Gamma; B \rrbracket}$$

Value Terms

$$\frac{\llbracket \Gamma, x : A \rrbracket}{\llbracket \Gamma, x : A; x \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma, x : A \rrbracket \arrow[d, "\eta"] \\ \pi^*_{\llbracket \Gamma; A \rrbracket} \coprod_{\llbracket \Gamma; A \rrbracket} 1\llbracket \Gamma, x : A \rrbracket \arrow[d, "\pi^*_{\llbracket \Gamma; A \rrbracket}\mathbf{fst}"] \\ \pi^*_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma; A \rrbracket \end{tikzcd}\right)}$$

$$\frac{\llbracket \Gamma_1, x : A, \Gamma_2; B \rrbracket \qquad \llbracket \Gamma_1, x : A, \Gamma_2; x \rrbracket : 1\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket \to A}{\llbracket \Gamma_1, x : A, \Gamma_2; y : B; x \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma_1, x : A, \Gamma_2, y : B \rrbracket \arrow[d, equal] \\ \pi^*_{\llbracket \Gamma_1, x:A, \Gamma_2; B \rrbracket} 1\llbracket \Gamma_1, x : A, \Gamma_2 \rrbracket \arrow[d, "\pi^*_{\llbracket \Gamma_1, x:A, \Gamma_2; B \rrbracket} \llbracket \Gamma_1, x:A, \Gamma_2; x \rrbracket"] \\ \pi^*_{\llbracket \Gamma_1, x:A, \Gamma_2; B \rrbracket} A \end{tikzcd}\right)}$$

$$\frac{\llbracket \Gamma \rrbracket \in \mathbb{B}}{\llbracket \Gamma; * \rrbracket = \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\mathrm{id}_{1\llbracket \Gamma \rrbracket}"] \\ 1\llbracket \Gamma \rrbracket \end{tikzcd}\right)}$$

$$\frac{\llbracket \Gamma \rrbracket \qquad \llbracket c \rrbracket : 1 \to \llbracket \diamond; \mathrm{ty}(c) \rrbracket}{\llbracket \Gamma; c \rrbracket = \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "!^* \llbracket c \rrbracket"] \\ !^* \llbracket \diamond; \mathrm{ty}(c) \rrbracket \end{tikzcd}\right)}$$

$$\frac{\llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to \llbracket \Gamma; A \rrbracket \qquad \llbracket \Gamma; W \rrbracket : 1\llbracket \Gamma \rrbracket \to (s\llbracket \Gamma; V \rrbracket)^* \llbracket \Gamma, x : A; B \rrbracket}{\llbracket \Gamma; \langle V, W \rangle_{(x:A).B} \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\llbracket \Gamma; W \rrbracket"] \\ (s\llbracket \Gamma; V \rrbracket)^* \llbracket \Gamma, x : A; B \rrbracket \arrow[d, "(s\llbracket \Gamma; V \rrbracket)^* \eta"] \\ (s\llbracket \Gamma; V \rrbracket)^* \pi^*_{\llbracket \Gamma; A \rrbracket} \coprod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; B \rrbracket \arrow[d, equal] \\ \coprod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; B \rrbracket \end{tikzcd}\right)}$$

$$\frac{\llbracket \Gamma; M \rrbracket : 1\llbracket \Gamma \rrbracket \to U\underline{C}}{\llbracket \Gamma; \mathbf{thunk}\ M \rrbracket = \llbracket \Gamma; M \rrbracket}$$

$$\frac{\llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to \llbracket \Gamma; A \rrbracket \qquad \llbracket \Gamma; B \rrbracket}{\llbracket \Gamma; \mathbf{inl}_{A+B}\ V \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\llbracket \Gamma; V \rrbracket"] \\ \llbracket \Gamma; A \rrbracket \arrow[d, "\iota_1"] \\ \llbracket \Gamma; A \rrbracket + \llbracket \Gamma; B \rrbracket \end{tikzcd}\right)}$$

$$\frac{\llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to \llbracket \Gamma; B \rrbracket \qquad \llbracket \Gamma; A \rrbracket}{\llbracket \Gamma; \mathbf{inr}_{A+B}\ V \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\llbracket \Gamma; V \rrbracket"] \\ \llbracket \Gamma; B \rrbracket \arrow[d, "\iota_2"] \\ \llbracket \Gamma; A \rrbracket + \llbracket \Gamma; B \rrbracket \end{tikzcd}\right)}$$

Computation Terms

$$\frac{\llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to A}{\llbracket \Gamma; \mathbf{return}\ V \rrbracket = \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\llbracket \Gamma; V \rrbracket"] \\ A \arrow[d, "\eta_A"] \\ UFA \end{tikzcd}\right)}$$

$$\frac{[\![\Gamma; M]\!] : 1[\![\Gamma]\!] \to UF[\![\Gamma; A]\!] \qquad [\![\Gamma, x : A; N]\!] : 1[\![\Gamma, x : A]\!] \to U\pi^*_{[\![\Gamma;A]\!]}[\![\Gamma; \underline{C}]\!]}{[\![\Gamma; M \textbf{ to } x : A \textbf{ in}_{\underline{C}}\ N]\!] \quad = \quad \left(\begin{tikzcd} 1[\![\Gamma]\!] \arrow[d, "{[\![\Gamma;M]\!]}"] \\ UF[\![\Gamma; A]\!] \arrow[d, "{UF\phi([\![\Gamma,x:A;N]\!])}"] \\ UFU[\![\Gamma; \underline{C}]\!] \arrow[d, "U\epsilon"] \\ U[\![\Gamma; \underline{C}]\!] \end{tikzcd}\right)}$$

Here, the isomorphism

$$\phi : \mathbb{E}_{\{X\}}(1\{[\![\Gamma; A]\!]\}, \pi^*_{[\![\Gamma;A]\!]}U[\![\Gamma; \underline{C}]\!]) \cong \mathbb{E}_I(\{[\![\Gamma; A]\!]\}, U[\![\Gamma; \underline{C}]\!])$$

is defined in Section 2.1.1.

$$\frac{[\![\Gamma; V]\!] : 1[\![\Gamma]\!] \to U[\![\Gamma; \underline{C}]\!]}{[\![\Gamma; \textbf{force}_{\underline{C}}\ V]\!] = \left(\begin{tikzcd} 1[\![\Gamma]\!] \arrow[d, "{[\![\Gamma;V]\!]}"] \\ U[\![\Gamma; \underline{C}]\!] \end{tikzcd}\right)}$$

$$\frac{\begin{array}{c} [\![\Gamma; V]\!] : 1[\![\Gamma]\!] \to \coprod_{[\![\Gamma;A]\!]} [\![\Gamma, x : A; B]\!] \\ [\![\Gamma, x : A, y : B; M]\!] : 1[\![\Gamma, x : A, y : B]\!] \to U\kappa^*[\![\Gamma, z : \Sigma x{:}A.B; \underline{C}]\!] \end{array}}{\begin{array}{l} [\![\Gamma; \textbf{pm}\ V \textbf{ as } \langle x : A, y : B\rangle \textbf{ in}_{z.\underline{C}}\ M]\!] \\ \qquad = \left(\begin{tikzcd} 1[\![\Gamma]\!] \arrow[d, equal] \\ (s[\![\Gamma; V]\!])^*(\kappa^{-1})^*1[\![\Gamma, x : A, y : B]\!] \arrow[d, "{(s[\![\Gamma;V]\!])^*(\kappa^{-1})^*[\![\Gamma,x:A,y:B;M]\!]}"] \\ (s[\![\Gamma; V]\!])^*(\kappa^{-1})^*U\kappa^*[\![\Gamma, z : \Sigma x{:}A.B; \underline{C}]\!] \arrow[d, equal] \\ U(s[\![\Gamma; V]\!])^*[\![\Gamma, z : \Sigma x{:}A.B; \underline{C}]\!] \end{tikzcd}\right) \end{array}}$$

$$\dfrac{\llbracket \Gamma, x : A; M \rrbracket : 1\llbracket \Gamma, x : A \rrbracket \to U\underline{C}}{\llbracket \Gamma; \lambda x{:}A.M \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\eta"] \\ \prod_{\llbracket \Gamma; A \rrbracket} \pi^*_{\llbracket \Gamma; A \rrbracket} 1\llbracket \Gamma \rrbracket \arrow[d, equal] \\ \prod_{\llbracket \Gamma; A \rrbracket} 1\llbracket \Gamma, x : A \rrbracket \arrow[d, "\prod \llbracket \Gamma, x:A; M \rrbracket"] \\ \prod_{\llbracket \Gamma; A \rrbracket} U\underline{C} \arrow[d, "\zeta^{-1}_{\llbracket \Gamma; A \rrbracket}"] \\ U \prod_{\llbracket \Gamma; A \rrbracket} \underline{C} \end{tikzcd}\right)}$$

Here, $\zeta_{\llbracket \Gamma; A \rrbracket} : U \prod_{\llbracket \Gamma; A \rrbracket} \underline{C} \to \prod_{\llbracket \Gamma; A \rrbracket} U\underline{C}$ is the isomorphism induced by the uniqueness of right adjoints of $\pi^*_{\llbracket \Gamma; A \rrbracket} F = F\pi^*_{\llbracket \Gamma; A \rrbracket}$ [9, Proposition 4.1.14].

$$\dfrac{\llbracket \Gamma; M \rrbracket : 1\llbracket \Gamma \rrbracket \to U \prod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; \underline{C} \rrbracket \qquad \llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to \llbracket \Gamma; A \rrbracket}{\llbracket \Gamma; M(V)_{(x:A).\underline{C}} \rrbracket \quad = \quad \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, "\llbracket \Gamma; M \rrbracket"] \\ U \prod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; \underline{C} \rrbracket \arrow[d, equal] \\ U(s\llbracket \Gamma; V \rrbracket)^* \pi^*_{\llbracket \Gamma; A \rrbracket} \prod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; \underline{C} \rrbracket \arrow[d, "U(s\llbracket \Gamma; V \rrbracket)^* \epsilon"] \\ U(s\llbracket \Gamma; V \rrbracket)^* \llbracket \Gamma, x : A; \underline{C} \rrbracket \end{tikzcd}\right)}$$

$$\frac{\begin{array}{c}\llbracket \Gamma; V \rrbracket : 1\llbracket \Gamma \rrbracket \to \llbracket \Gamma; A \rrbracket + \llbracket \Gamma; B \rrbracket \\ \llbracket \Gamma, x : A; M \rrbracket : 1\llbracket \Gamma, x : A \rrbracket \to U\{\iota_1\}^* \llbracket \Gamma, z : A + B; \underline{C} \rrbracket \\ \llbracket \Gamma, y : B; N \rrbracket : 1\llbracket \Gamma, y : B \rrbracket \to U\{\iota_2\}^* \llbracket \Gamma, z : A + B; \underline{C} \rrbracket\end{array}}{\llbracket \Gamma; \textbf{case } V \textbf{ of}_{z.\underline{C}} \ (\textbf{inl } (x : A) \mapsto M, \textbf{inr } (y : B) \mapsto N) \rrbracket}$$

$$= \left(\begin{tikzcd} 1\llbracket \Gamma \rrbracket \arrow[d, equal] \\ (s\llbracket \Gamma; V \rrbracket)^* 1\llbracket \Gamma, z : A + B \rrbracket \arrow[d, "(s\llbracket \Gamma;V \rrbracket)^* [\llbracket \Gamma, x:A; M \rrbracket, \llbracket \Gamma, y:B; N \rrbracket]"] \\ (s\llbracket \Gamma; V \rrbracket)^* U\llbracket \Gamma, z : A + B; \underline{C} \rrbracket \arrow[d, equal] \\ U(s\llbracket \Gamma; V \rrbracket)^* \llbracket \Gamma, z : A + B; \underline{C} \rrbracket \end{tikzcd} \right)$$

A.2 Refinement Type System

We list the full definition of our dependent refinement type system.

A.2.1 Typing Rules

The main differences in typing rules from the underlying type system are that we use refinement types $\{v : b(V) \mid p\}$ instead of base types $b(V)$ and that we use subtypings $\Gamma \vdash A <: B$ instead of type equalities $\Gamma \vdash A = B$. Note that terms remain the same as the underlying type system.

Well-Formed Contexts

$$\frac{}{\vdash \diamond} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A \qquad x \notin \mathrm{Vars}(\Gamma)}{\vdash \Gamma, x : A}$$

Context Subtyping

$$\frac{}{\vdash \diamond <: \diamond}$$

$$\frac{\vdash \Gamma_1 <: \Gamma_2 \qquad \Gamma_1 \vdash A <: B \qquad \Gamma_2 \vdash B \qquad x \notin \mathrm{Vars}(\Gamma_1) \cup \mathrm{Vars}(\Gamma_2)}{\vdash \Gamma_1, x : A <: \Gamma_2, x : B}$$

Subtyping

Reflexivity.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A <: A} \qquad \frac{\Gamma \vdash \underline{C}}{\Gamma \vdash \underline{C} <: \underline{C}}$$

Transitivity.

$$\frac{\Gamma \vdash A_1 <: A_2 \qquad \Gamma \vdash A_2 <: A_3}{\Gamma \vdash A_1 <: A_3} \qquad \frac{\Gamma \vdash \underline{C}_1 <: \underline{C}_2 \qquad \Gamma \vdash \underline{C}_2 <: \underline{C}_3}{\Gamma \vdash \underline{C}_1 <: \underline{C}_3}$$

Subsumption

$$\frac{\Gamma_2 \vdash A \qquad \vdash \Gamma_1 <: \Gamma_2}{\Gamma_1 \vdash A} \qquad\qquad \frac{\Gamma_2 \vdash \underline{C} \qquad \vdash \Gamma_1 <: \Gamma_2}{\Gamma_1 \vdash \underline{C}}$$

$$\frac{\Gamma_2 \vdash A <: B \qquad \vdash \Gamma_1 <: \Gamma_2}{\Gamma_1 \vdash A <: B} \qquad\qquad \frac{\Gamma_2 \vdash \underline{C} <: \underline{D} \qquad \vdash \Gamma_1 <: \Gamma_2}{\Gamma_1 \vdash \underline{C} <: \underline{D}}$$

$$\frac{\vdash \Gamma_1 <: \Gamma_2 \qquad \Gamma_2 \vdash V \ : \ A \qquad \Gamma_1 \vdash A <: B}{\Gamma_1 \vdash V \ : \ B}$$

$$\frac{\vdash \Gamma_1 <: \Gamma_2 \qquad \Gamma_2 \vdash M \ : \ \underline{C} \qquad \Gamma_1 \vdash \underline{C} <: \underline{D}}{\Gamma_1 \vdash M \ : \ \underline{D}}$$

Variables

$$\frac{\vdash \Gamma_1, x : A, \Gamma_2}{\Gamma_1, x : A, \Gamma_2 \vdash x \ : \ A} \qquad\qquad \frac{\vdash \Gamma_1, x : \{v : b(V) \mid p\}, \Gamma_2}{\Gamma_1, x : \{v : b(V) \mid p\}, \Gamma_2 \vdash x \ : \ \{v : b(V) \mid v = x\}}$$

Value constants

$$\frac{\vdash \Gamma \qquad \diamond \vdash \mathrm{ty}(c)}{\Gamma \vdash c_{|\mathrm{ty}(c)|} \ : \ \mathrm{ty}(c)}$$

Unit Type

$$\frac{\vdash \Gamma}{\Gamma \vdash * \ : \ \{v : 1 \mid \top\}} \qquad\qquad \frac{\vdash \Gamma \qquad |\Gamma|, v : 1 \vdash p : \mathrm{Prop}}{\Gamma \vdash \{v : 1 \mid p\}}$$

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

Refinement Types

$$\frac{\vdash \Gamma \qquad b : A_u \to \mathrm{Type} \qquad |\Gamma| \vdash b(V) \qquad |\Gamma|, v : b(V) \vdash p : \mathrm{Prop}}{\Gamma \vdash \{v : b(V) \mid p\}}$$

$$\frac{\vdash \Gamma \qquad |\Gamma| \vdash b(V) = b(W) \qquad \Gamma; v : b(V) \mid p \vdash q}{\Gamma \vdash \{v : b(V) \mid p\} <: \{v : b(W) \mid q\}}$$

Value Σ-Types

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Sigma x{:}A.B} \qquad\qquad \frac{\Gamma \vdash V \ : \ A \qquad \Gamma, x : A \vdash B \qquad \Gamma \vdash W \ : \ B[V/x]}{\Gamma \vdash \langle V, W \rangle \ : \ \Sigma x{:}A.B}$$

$$\frac{\Gamma \vdash V \ : \ \Sigma x{:}A.B \qquad \Gamma, z : \Sigma x{:}A.B \vdash \underline{C} \qquad \Gamma, x : A, y : B \vdash M \ : \ \underline{C}[\langle x, y \rangle / z]}{\Gamma \vdash \mathbf{pm}\ V\ \mathbf{as}\ \langle x : |A|, y : |B| \rangle\ \mathbf{in}_{z.|\underline{C}|}\ M \ : \ \underline{C}[V/z]}$$

$$\frac{\Gamma \vdash A_1 <: A_2 \qquad \Gamma, x : A_2 \vdash B_2 \qquad \Gamma, x : A_1 \vdash B_1 <: B_2}{\Gamma \vdash \Sigma x : A_1.B_1 <: \Sigma x : A_2.B_2}$$

Thunked Computation

$$\frac{\Gamma \vdash \underline{C}}{\Gamma \vdash U\underline{C}} \qquad \frac{\Gamma \vdash M \ : \ \underline{C}}{\Gamma \vdash \textbf{thunk}\ M \ : \ U\underline{C}} \qquad \frac{\Gamma \vdash V \ : \ U\underline{C}}{\Gamma \vdash \textbf{force}_{|\underline{C}|}\ V \ : \ \underline{C}}$$

$$\frac{\Gamma \vdash \underline{C}_1 <: \underline{C}_2}{\Gamma \vdash U\underline{C}_1 <: U\underline{C}_2}$$

Return

$$\frac{\Gamma \vdash A}{\Gamma \vdash FA} \qquad \frac{\Gamma \vdash V \ : \ A}{\Gamma \vdash \textbf{return}\ V \ : \ FA}$$

$$\frac{\Gamma \vdash M \ : \ FA \qquad \Gamma \vdash \underline{C} \qquad \Gamma, x : A \vdash N \ : \ \underline{C}}{\Gamma \vdash M\ \textbf{to}\ x : |A|\ \textbf{in}_{|\underline{C}|}\ N \ : \ \underline{C}} \qquad \frac{\Gamma \vdash A_1 <: A_2}{\Gamma \vdash FA_1 <: FA_2}$$

Computational Π-Types

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash \underline{C}}{\Gamma \vdash \Pi x{:}A.\underline{C}}$$

$$\frac{\Gamma \vdash A_2 <: A_1 \qquad \Gamma, x : A_1 \vdash \underline{C}_1 \qquad \Gamma, x : A_2 \vdash \underline{C}_1 <: \underline{C}_2}{\Gamma \vdash \Pi x{:}A_1.\underline{C}_1 <: \Pi x{:}A_2.\underline{C}_2}$$

$$\frac{\Gamma, x : A \vdash M \ : \ \underline{C}}{\Gamma \vdash \lambda x : |A|.M \ : \ \Pi x{:}A.\underline{C}}$$

$$\frac{\Gamma, x : A \vdash \underline{C} \qquad \Gamma \vdash M \ : \ \Pi x{:}A.\underline{C} \qquad \Gamma \vdash V \ : \ A}{\Gamma \vdash M(V)_{(x:|A|).|\underline{C}|} \ : \ \underline{C}[V/x]}$$

Fibred Coproduct Types

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A + B} \qquad \frac{\Gamma \vdash V \ : \ A \qquad \Gamma \vdash B}{\Gamma \vdash \textbf{inl}_{A+B}\ V \ : \ A + B} \qquad \frac{\Gamma \vdash V \ : \ B \qquad \Gamma \vdash A}{\Gamma \vdash \textbf{inr}_{A+B}\ V \ : \ A + B}$$

$$\frac{\begin{array}{c}\Gamma, z : A + B \vdash \underline{C} \qquad \Gamma \vdash V \ : \ A + B \\ \Gamma, x : A \vdash M \ : \ \underline{C}[\textbf{inl}_{A+B}\ x/z] \qquad \Gamma, y : B \vdash N \ : \ \underline{C}[\textbf{inr}_{A+B}\ y/z]\end{array}}{\Gamma \vdash \textbf{case}\ V\ \textbf{of}_{z.\underline{C}}\ (\textbf{inl}\ (x : A) \mapsto M, \textbf{inr}\ (y : B) \mapsto N) \ : \ \underline{C}[V/z]}$$

$$\frac{\Gamma \vdash A_1 <: A_2 \qquad \Gamma \vdash B_1 <: B_2}{\Gamma \vdash A_1 + B_1 <: A_2 + B_2}$$

A.2.2 Semantics

Given a lifting of a fibred adjunction model (Definition 2.4.6), we define the interpretation of types in the dependent refinement type system as follows. Note that the interpretation of a term is the same as the underlying type system. The most essential part of the definition is the interpretation $[\![\Gamma; \{v : b(V) \mid p\}]\!]$ (and $[\![\Gamma; \{v : 1 \mid p\}]\!]$) of refinement types.

Contexts

$$\frac{}{\llbracket \diamond \rrbracket = 1 \in \mathbb{P}_1} \qquad \frac{\llbracket \Gamma; A \rrbracket \in \{\mathbb{E} \mid \mathbb{P}\}_{\llbracket \Gamma \rrbracket} \qquad x \notin \mathrm{Vars}(\Gamma)}{\llbracket \Gamma, x : A \rrbracket = \{\llbracket \Gamma; A \rrbracket\}}$$

Types

$$\frac{\llbracket \Gamma \rrbracket \in \mathbb{P}_I \qquad \llbracket |\Gamma|; b(V) \rrbracket \in \mathbb{E}_I \qquad \llbracket |\Gamma|, v : b(V) \vdash p \rrbracket \in \mathbb{P}_{\{X\}}}{\llbracket \Gamma; \{v : b(V) \mid p\} \rrbracket = \big(\llbracket |\Gamma|; b(V) \rrbracket, \llbracket \Gamma \rrbracket, \pi^*_{\llbracket |\Gamma|; b(V) \rrbracket} \llbracket \Gamma \rrbracket \wedge \llbracket |\Gamma|, v : b(V) \vdash p \rrbracket\big)}$$

$$\frac{\llbracket \Gamma \rrbracket \in \mathbb{P} \qquad \llbracket |\Gamma|, v : 1 \vdash p \rrbracket \in \mathbb{P}_{\{1 q \llbracket \Gamma \rrbracket\}}}{\llbracket \Gamma; \{v : 1 \mid p\} \rrbracket = \big(1 q \llbracket \Gamma \rrbracket, \llbracket \Gamma \rrbracket, \pi^*_{1 q \llbracket \Gamma \rrbracket} \llbracket \Gamma \rrbracket \wedge \llbracket |\Gamma|, v : 1 \vdash p \rrbracket\big)}$$

$$\frac{\llbracket \Gamma; A \rrbracket \qquad \llbracket \Gamma, x : A; B \rrbracket}{\llbracket \Gamma; \Sigma x{:}A.B \rrbracket = \coprod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; B \rrbracket} \qquad \frac{\llbracket \Gamma; \underline{C} \rrbracket}{\llbracket \Gamma; U\underline{C} \rrbracket = \dot{U} \llbracket \Gamma; \underline{C} \rrbracket}$$

$$\frac{\llbracket \Gamma; A \rrbracket}{\llbracket \Gamma; FA \rrbracket = \dot{F} \llbracket \Gamma; A \rrbracket} \qquad \frac{\llbracket \Gamma; A \rrbracket \qquad \llbracket \Gamma, x : A; \underline{C} \rrbracket}{\llbracket \Gamma; \Pi x{:}A.\underline{C} \rrbracket = \prod_{\llbracket \Gamma; A \rrbracket} \llbracket \Gamma, x : A; \underline{C} \rrbracket}$$

Appendix B

Details of Chapter 5

B.1 Preliminaries on Measure Theory

In this section, we review some results from measure theory that is needed in the rest of the chapter. For more details, see e.g. [13, 113].

Definition B.1.1. Let $\phi : X \to Y$ be a measurable function and μ be a probability measure on X. A *pushforward measure* $(\phi)_*\mu$ is a measure on Y defined by $(\phi)_*\mu(E) = \mu(\phi^{-1}(E))$ for each measurable set $E \subseteq Y$.

Lemma B.1.2. Let $\phi : X \to Y$ and $f : Y \to [0, \infty]$ be measurable functions and μ be a probability measure on X.

$$\int f \,\mathrm{d}\big((\phi)_*\mu\big) = \int (f \circ \phi) \,\mathrm{d}\mu$$

where $f \circ \phi$ denotes the composite function of f and ϕ. □

Lemma B.1.3. Let (X, B_X) and (Y, B_Y) be measurable spaces and μ_x be a probability measure on Y for each $x \in X$. The following conditions are equivalent.

1. For each $E \in B_Y$, a mapping $x \mapsto \mu_x(E)$ is measurable.
2. For each measurable function $f : X \times Y \to [0, \infty]$,

   $$x \mapsto \int_Y f(x, y) d\mu_x(y)$$

   is measurable.

Proof.

(1 $\implies$ 2) We write $B_{X\times Y}$ for the product σ-algebra of B_X and B_Y. By the monotone convergence theorem (see e.g. [13, Theorem 1.6.2]) and the linearity of integration, it suffices to prove that for each $E \in B_{X\times Y}$, $f = 1_E : X \times Y \to [0, \infty]$ satisfies the condition 2. Let $M = \{E \in B_{X\times Y} \mid$

1_E satisfies the condition 2$\}$. By the monotone class theorem (see e.g. [13, Theorem 1.3.9]), to prove $M = B_{X\times Y}$, it suffices to prove that M is a monotone class and contains a Boolean algebra

$$\{\bigcup_{i=1}^{n}(E_i \times F_i) \mid E_i \in B_X, F_i \in B_Y\}.$$

The rest of the proof is easy.

(2 $\implies$ 1) Given $E \in B_Y$, consider $f = 1_{X\times E}$. □

For any $f : A \to B$ and any set X, $X^f : X^B \to X^A$ denotes a precomposition of f i.e. $X^f(u) = u \circ f$. If $A \subseteq B$, we write $X^{A\subseteq B}$ for $X^i : X^B \to X^A$ where $i : A \to B$ is the inclusion mapping.

In Section B.2, we use the following corollary of the Kolmogorov extension theorem (see [113, Section 2.4]).

Corollary B.1.4. Let (X, B_X) be a measurable space and μ_n be an inner regular probability measure on X^n for each $n < \omega$. Assume $(X^{n\subseteq n+1})_*\mu_{n+1} = \mu_n$. There exists a unique probability measure μ_ω on X^ω such that $(X^{n\subseteq\omega})_*\mu_\omega = \mu_n$. □

B.2 Higher Moments of Runtimes and Rewards

We define a probably measure on the set of runs of a pCFG given a scheduler. We then define the k-th moment of runtimes. Here we slightly generalize runtime model by considering a reward function and redefine some of the notions to accommodate the reward function. However, this generalization is not essential, and therefore the readers can safely assume that we are just counting the number of steps until termination (by taking the constant function 1 as a reward function).

Let $\Gamma = (L, V, l_{\text{init}}, \vec{x}_{\text{init}}, \mapsto, \text{Up}, \text{Pr}, G)$ be a pCFG. A *reward function* on Γ is a measurable function $\text{Rew} : S \to [0, \infty]$. Recall that we regard the set $S = L \times \mathbb{R}^V$ of configurations as the product measurable space of $(L, 2^L)$ and $(\mathbb{R}^V, \mathcal{B}(\mathbb{R}^V))$. A *scheduler* of Γ resolves two types of nondeterminism: nondeterministic transition and nondeterministic assignment.

Definition B.2.1 (scheduler)**.** A *scheduler* $\sigma = (\sigma_t, \sigma_a)$ of Γ consists of the following components.

- A function $\sigma_t : (L \times \mathbb{R}^V)^*(L_N \times \mathbb{R}^V) \to \mathcal{D}(L)$ such that
  - if $\pi \in (L \times \mathbb{R}^V)^*(L_N \times \mathbb{R}^V)$ and $l \in L_N$ is the last location of π, then $l' \in \text{supp}(\sigma_t(\pi))$ implies $l \mapsto l'$, and
  - for each $l \in L$, the mapping $\pi \mapsto \sigma_t(\pi)(\{l\}) : (L \times \mathbb{R}^V)^*(L_N \times \mathbb{R}^V) \to [0, 1]$ is measurable.
- A function $\sigma_a : (L \times \mathbb{R}^V)^*(L_{AN} \times \mathbb{R}^V) \to \mathcal{D}(\mathbb{R})$ such that
  - if $\pi \in (L \times \mathbb{R}^V)^*(L_{AN} \times \mathbb{R}^V)$, $l \in L_{AN}$ is the last location of π and $(x_j, u) = \text{Up}(l)$, then $\text{supp}(\sigma_a(\pi)) \subseteq u$, and

  - for each $A \in \mathcal{B}(\mathbb{R})$, the mapping $\pi \mapsto \sigma_a(\pi)(A)$ is measurable.

Note that if $L_N = \emptyset$ and $L_{AN} = \emptyset$, then there exists only one scheduler that is trivial.

In the rest of the chapter, the concatenation of two finite sequences ρ_1, ρ_2 is denoted by $\rho_1\rho_2$ or by $\rho_1 \cdot \rho_2$.

Given a scheduler σ and a history of configurations $\rho \in S^+$, let μ^σ_ρ be a probability distribution of the next configurations determined by σ.

Definition B.2.2. Let σ be a scheduler and $\rho \in S^+$. A probability measure μ^σ_ρ on S is defined as follows.

- If $l \in L_D$ and $\vec{x} \vDash G(l, l')$, $\mu^\sigma_{\rho\cdot(l,\vec{x})} = \delta_{(l',\vec{x})}$.
- If $l \in L_P$, $\mu^\sigma_{\rho\cdot(l,\vec{x})} = \sum_{l\mapsto l'} \mathrm{Pr}_l(l')\delta_{(l',\vec{x})}$.
- If $l \in L_N$, $\mu^\sigma_{\rho\cdot(l,\vec{x})} = \sum_{l\mapsto l'} \sigma_t(\rho \cdot (l, \vec{x}))(\{l'\})\delta_{(l',\vec{x})}$.
- Assume $l \in L_A$, $\mathrm{Up}(l) = (x_j, u)$ and $l \mapsto l'$.
  - If $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$, $\mu^\sigma_{\rho\cdot(l,\vec{x})} = \delta_{(l',\vec{x}(x_j\leftarrow u(\vec{x})))}$.
  - If $u \in \mathcal{D}(\mathbb{R})$, $\mu^\sigma_{\rho\cdot(l,\vec{x})} = (\lambda y.(l', \vec{x}(x_j \leftarrow y)))_* u$.
  - If $u \in \mathcal{B}(\mathbb{R})$, $\mu^\sigma_{\rho\cdot(l,\vec{x})} = (\lambda y.(l', \vec{x}(x_j \leftarrow y)))_* \sigma_a(\rho \cdot (l, \vec{x}))$.

Lemma B.2.3. For each $E \in \mathcal{B}(S)$, a mapping $\rho \mapsto \mu^\sigma_\rho(E) : S^+ \to [0, 1]$ is measurable.

Proof. Let $f : S^+ \to [0, 1]$ be a function defined by $f(\rho) = \mu^\sigma_\rho(E)$. It suffices to prove that for each $n < \omega$ and $l \in L$, $f|_{S^n\times(\{l\}\times\mathbb{R}^V)} : S^n \times (\{l\} \times \mathbb{R}^V) \to [0, 1]$ is measurable, that is, a function $g_{n,l} : S^n \times \mathbb{R}^V \to [0, 1]$ defined by $g_{n,l}(\rho, \vec{x}) = \mu^\sigma_{\rho\cdot(l,\vec{x})}(E)$ is measurable.

- Assume $l \in L_D$.
$$g_{n,l}(\rho, \vec{x}) = \delta_{(l,\vec{x})}(E) = 1_E(l, \vec{x})$$
- Assume $l \in L_P$.
$$g_{n,l}(\rho, \vec{x}) = \sum_{l\mapsto l'} \mathrm{Pr}_l(l')1_E(l', \vec{x})$$
- Assume $l \in L_N$.
$$g_{n,l}(\rho, \vec{x}) = \sum_{l\mapsto l'} \sigma_t(\rho \cdot (l, \vec{x}))(\{l'\})1_E(l', \vec{x})$$
- Assume $l \in L_A$, $\mathrm{Up}(l) = (x_j, u)$ and $l \mapsto l'$.
  - Assume $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$.
$$g_{n,l}(\rho, \vec{x}) = 1_E(l', \vec{x}(x_j \leftarrow u(\vec{x})))$$
  - Assume $u \in \mathcal{D}(\mathbb{R})$.
$$g_{n,l}(\rho, \vec{x}) = \int_{\mathbb{R}} 1_E(l', \vec{x}(x_j \leftarrow y))\,\mathrm{d}u(y)$$

– Assume $u \in \mathcal{B}(\mathbb{R})$.

$$g_{n,l}(\rho, \vec{x}) = \int_{\mathbb{R}} 1_E(l', \vec{x}(x_j \leftarrow y)) \,\mathrm{d}(\sigma_a(\rho \cdot (l, \vec{x})))(y)$$

In each case, it easily follows that $g_{n,l}$ is measurable. Note that $\delta_{(-)}(E) = 1_E$ and $(\vec{x}, y) \mapsto \vec{x}(x_j \leftarrow y)$ are measurable functions. We use Lemma B.1.3 for the last two cases. □

Given an initial configuration c_0, let $\nu^\sigma_{c_0,n}$ be a probability measure on the set $\{c_0\rho \mid \rho \in S^n\} \cong S^n$ of paths.

Definition B.2.4. For each $n \in \omega$, $\nu^\sigma_{c_0,n}$ is a probability measure on S^n defined as

$$\nu^\sigma_{c_0,n}(E) = \begin{cases} \int_S \cdots \int_S 1_E(c_1, \ldots, c_n) \,\mathrm{d}\mu^\sigma_{c_0 \ldots c_{n-1}}(c_n) \ldots \mathrm{d}\mu_{c_0}(c_1) & \text{if } n > 0 \\ \delta_* & \text{if } n = 0 \end{cases}$$

where $*$ is the element of $S^0 = \{*\}$.

Definition B.2.4 is well-defined by Lemma B.1.3 and Lemma B.2.3.

The following lemma is a fundamental property of $\nu^\sigma_{c_0,n}$.

Lemma B.2.5. Assume $n > 0$. For any measurable function $f : S^n \to [0, \infty]$,

$$\int f \,\mathrm{d}\nu^\sigma_{c_0,n} = \int_S \cdots \int_S f(c_1, \ldots, c_n) \,\mathrm{d}\mu^\sigma_{c_0 \ldots c_{n-1}}(c_n) \ldots \mathrm{d}\mu^\sigma_{c_0}(c_1).$$

Proof. By the monotone convergence theorem and the linearity of integration. □

Lemma B.2.6. For each $n \in \mathbb{N}$, $(S^{n \subseteq n+1})_* \nu^\sigma_{c_0,n+1} = \nu^\sigma_{c_0,n}$.

Proof.

$$\begin{aligned}
&((S^{n \subseteq n+1})_* \nu^\sigma_{c_0,n+1})(E) \\
&= \nu^\sigma_{c_0,n+1}(E \times S) \\
&= \int_S \cdots \int_S 1_{E \times S}(c_1, \ldots, c_{n+1}) \,\mathrm{d}\mu^\sigma_{c_0 \ldots c_n}(c_{n+1}) \ldots \mathrm{d}\mu^\sigma_{c_0}(c_1) \\
&= \int_S \cdots \int_S \left(\int_S 1_S(c_{n+1}) \,\mathrm{d}\mu^\sigma_{c_0 \ldots c_n}(c_{n+1}) \right) \cdot 1_E(c_1, \ldots, c_n) \,\mathrm{d}\mu^\sigma_{c_0 \ldots c_{n-1}}(c_n) \ldots \mathrm{d}\mu^\sigma_{c_0}(c_1) \\
&= \int_S \cdots \int_S 1_E(c_1, \ldots, c_n) \,\mathrm{d}\mu^\sigma_{c_0 \ldots c_{n-1}}(c_n) \ldots \mathrm{d}\mu^\sigma_{c_0}(c_1) \\
&= \nu^\sigma_{c_0,n}(E)
\end{aligned}$$

□

By Corollary B.1.4, we define a probability measure on S^ω. Note that $(S, \mathcal{B}(S))$ is a Polish space (a separable completely metrizable topological space), and hence a Radon space. Therefore, $\nu^\sigma_{c_0,n}$ is inner regular.

Definition B.2.7. Let $\nu^\sigma_{c_0}$ be the probability measure defined as a unique measure such that $(S^{n \subseteq \omega})_* \nu^\sigma_{c_0} = \nu^\sigma_{c_0,n}$.

Definition B.2.8 (accumulated reward $\mathrm{Rew}_C^{c_0}$)**.** Given a reward function $\mathrm{Rew} : S \to [0,\infty]$, let $\mathrm{Rew}_C^{c_0} : S^\omega \to [0,\infty]$ be a measurable function defined by the sum of the rewards from the initial configuration c_0 to the last configuration before entering C. That is,

$$\mathrm{Rew}_C^{c_0}(c_1c_2\dots) = \begin{cases} \sum_{j=0}^{N-1} \mathrm{Rew}(c_j) & \exists N \geq 0 \text{ s.t. } c_N \in C \land (0 \leq j < N \implies c_j \notin C) \\ \sum_{j=0}^{\infty} \mathrm{Rew}(c_j) & \text{otherwise (i.e. for each } i,\ c_i \notin C). \end{cases}$$

Note that $\mathrm{Rew}(c_0)$ is included in the sum.

Definition B.2.9 (k-th moment of rewards)**.** We define two functions $\mathbb{M}_{C,\sigma}^{\mathrm{Rew},k}, \overline{\mathbb{M}}_C^{\mathrm{Rew},k} : S \to [0,\infty]$ as follows.

$$\mathbb{M}_{C,\sigma}^{\mathrm{Rew},k}(c_0) = \int (\mathrm{Rew}_C^{c_0})^k \,\mathrm{d}\nu_{c_0}^{\sigma} \qquad \overline{\mathbb{M}}_C^{\mathrm{Rew},k}(c_0) = \sup_\sigma \mathbb{M}_{C,\sigma}^{\mathrm{Rew},k}(c_0)$$

Note that $\mathbb{M}_{C,\sigma}^{\mathrm{Rew},k}$ is measurable by Lemma B.1.3.

The correspondence of the notations in Section 5.1 and in Section B.2 is as follows.

Section 5.1	Section B.2
ν_σ^Γ	$\nu_{c_0}^\sigma$ where $c_0 = (l_{\mathrm{init}}, \vec{x}_{\mathrm{init}})$
$T_{C,\sigma}^\Gamma$	$\mathrm{Rew}_C^{c_0}$ where $c_0 = (l_{\mathrm{init}}, \vec{x}_{\mathrm{init}})$
$\mathbb{M}_{C,\sigma}^{\Gamma,k}$, $\overline{\mathbb{M}}_C^{\Gamma,k}$	$\mathbb{M}_{C,\sigma}^{\mathrm{Rew},k}$, $\overline{\mathbb{M}}_C^{\mathrm{Rew},k}$ where $\mathrm{Rew}(c) = 1$ for each c

B.3 Omitted Details and Proofs in Section 5.2

The ultimate goal of this section is to prove Theorem 5.2.13. In Section B.3.1-B.3.2, we prove some lemmas regarding to $\overline{\mathbb{X}}$ (Definition 5.2.1) and $\mathrm{El}_1^{K,k}$ (Definition 5.2.10). In Section B.3.3, we prove analogous theorem to Theorem 5.2.7, Theorem 5.2.9 and Theorem 5.2.13. In Section B.3.4, we prove Theorem 5.2.13. We prove them in a generalized way so that an arbitrary reward function is allowed as in Section B.2.

B.3.1 Basic Properties of the Pre-expectation

We prove several lemmas for $\overline{\mathbb{X}}$ in Definition 5.2.1.

The next lemma claims that we can ignore outside of an invariant I.

Lemma B.3.1. Let I be an invariant. If $\eta(c) = \eta'(c)$ for any $c \in I$, then $(\overline{\mathbb{X}}\eta)(c) = (\overline{\mathbb{X}}\eta')(c)$ for any $c \in I$. □

The complete lattice $[0,\infty]$ has the following properties as an ω-cpo, and the set of functions $S \to [0,\infty]^K$ inherits the same properties.

- Let $\{\eta_n\}_{n<\omega}$ and $\{\eta'_n\}_{n<\omega}$ be ω-chains. Then we have

  $$\sup_{n\in\omega} \eta_n + \sup_{n\in\omega} \eta'_n = \sup_{n\in\omega}(\eta_n + \eta'_n).$$

  That is, the addition $+$ is ω-continuous.

- Let $\{\eta_n\}_{n<\omega}$ be a ω-chain and $a \geq 0$. Then we have

$$a \cdot \sup_{n\in\omega} \eta_n = \sup_{n\in\omega}(a \cdot \eta_n).$$

  That is, $a \cdot (-)$ is ω-continuous.

These properties are often used in the proofs of ω-continuity in the rest of the chapter.

Lemma B.3.2. $\overline{\mathbb{X}}$ is ω-continuous.

Proof. Let $\{\eta_n : S \to [0, \infty]\}_{n\in\omega}$ be an ω-chain. We prove $(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) = \sup_{n\in\omega}(\overline{\mathbb{X}})(l, \vec{x})$ for each $(l, \vec{x}) \in L \times \mathbb{R}^V$.

- Assume $l \in L_D$ and $\vec{x} \vDash G(l, l')$.

$$(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) = (\sup_{n\in\omega} \eta_n)(l', \vec{x}) = \sup_{n\in\omega}(\eta_n(l', \vec{x})) = \sup_{n\in\omega}(\overline{\mathbb{X}}\eta_n)(l, \vec{x})$$

- Assume $l \in L_P$.

$$\begin{aligned}(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) &= \sum_{l\mapsto l'} \mathrm{Pr}_l(l')(\sup_{n\in\omega} \eta_n)(l', \vec{x}) \\ &= \sup_{n\in\omega} \sum_{l\mapsto l'} \mathrm{Pr}_l(l')\eta_n(l', \vec{x}) \\ &= \sup_{n\in\omega}(\overline{\mathbb{X}}\eta_n)(l, \vec{x})\end{aligned}$$

- Assume $l \in L_N$.

$$(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) = \sup_{l\mapsto l'} \sup_{n\in\omega} \eta_n(l', \vec{x}) = \sup_{n\in\omega} \sup_{l\mapsto l'} \eta_n(l', \vec{x}) = \sup_{n\in\omega}(\overline{\mathbb{X}}\eta_n)(l, \vec{x}).$$

- Assume $l \in L_A$, $\mathrm{Up}(l) = (x_j, u)$ and $l \mapsto l'$.
  - Assume $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$.

$$(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) = \sup_{n\in\omega} \eta_n(l', \vec{x}(x_j \leftarrow u(\vec{x}))) = \sup_{n\in\omega}(\overline{\mathbb{X}}\eta_n)(l, \vec{x})$$

  - Assume $u \in \mathcal{D}(\mathbb{R})$.

$$\begin{aligned}(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) &= \int_{\mathbb{R}} (\sup_{n\in\omega} \eta_n)(l', \vec{x}(x_j \leftarrow y))\, \mathrm{d}u(y) \\ &= \sup_{n\in\omega} \int_{\mathbb{R}} \eta_n(l', \vec{x}(x_j \leftarrow y))\, \mathrm{d}u(y) \\ &= \sup_{n\in\omega}(\overline{\mathbb{X}}\eta_n)(l, \vec{x})\end{aligned}$$

    by the monotone convergence theorem.

  - Assume $u \in \mathcal{B}(\mathbb{R})$.

$$\begin{aligned}(\overline{\mathbb{X}}(\sup_{n\in\omega} \eta_n))(l, \vec{x}) &= \sup_{y\in u} \sup_{n\in\omega} \eta_n(l', \vec{x}(x_j \leftarrow y)) \\ &= \sup_{n\in\omega} \sup_{y\in u} \eta_n(l', \vec{x}(x_j \leftarrow y)) \\ &= \sup_{n\in\omega}(\overline{\mathbb{X}}\eta_n)(l, \vec{x})\end{aligned}$$

□

The next lemma is a justification of the name “pre-expectation”.

Lemma B.3.3. For any configuration c_0, measurable function $\eta : S \to [0,\infty]$, scheduler σ,

$$\overline{\mathbb{X}}\eta(c_0) \geq \int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1).$$

Proof. Let $(l_0, \vec{x}_0) = c_0$.

- Assume $l_0 \in L_D$ and $\vec{x}_0 \vDash G(l_0, l_1)$.

$$\int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1) = \eta(l_1, \vec{x}_0) = \overline{\mathbb{X}}\eta(c_0)$$

- Assume $l_0 \in L_P$.

$$\int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1) = \sum_{l_0 \mapsto l_1} \mathrm{Pr}_{l_0}(l_1)\eta(l_1, \vec{x}_0) = \overline{\mathbb{X}}\eta(c_0)$$

- Assume $l_0 \in L_N$.

$$\int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1) = \sum_{l_0 \mapsto l_1} \sigma_t(c_0)(l_1)\eta(l_1, \vec{x}_0) \leq \sup_{l_0 \mapsto l_1} \eta(l_1, \vec{x}_0) = \overline{\mathbb{X}}\eta(c_0)$$

- Assume $l_0 \in L_A$, $\mathrm{Up}(l_0) = (x_j, u)$ and $l_0 \mapsto l_1$.
  - Assume $u \in \mathcal{B}(\mathbb{R}^V, \mathbb{R})$.

$$\int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1) = \eta(l_1, \vec{x}_0(x_j \leftarrow u(\vec{x}_0))) = \overline{\mathbb{X}}\eta(c_0)$$

  - Assume $u \in \mathcal{D}(\mathbb{R})$.

$$\int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1) = \int_{\mathbb{R}} \eta(l_1, \vec{x}_0(x_j \leftarrow y)) du(y) = \overline{\mathbb{X}}\eta(c_0)$$

  - Assume $u \in \mathcal{B}(\mathbb{R})$.

$$\begin{aligned}\int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1) &= \int_{\mathbb{R}} \eta(l_1, \vec{x}_0(x_j \leftarrow y)) d(\sigma_a(c_0))(y) \\ &\leq \sup_{y \in u} \eta(l_1, \vec{x}_0(x_j \leftarrow y)) = \overline{\mathbb{X}}\eta(c_0)\end{aligned}$$

□

Lemma B.3.4. Assume $L_N = \emptyset$ and $L_{AN} = \emptyset$ and let σ be the unique scheduler that plays no role. For any configuration c_0 and any measurable function $\eta : S \to [0,\infty]$,

$$\overline{\mathbb{X}}\eta(c_0) = \int_S \eta(c_1) d\mu^{\sigma}_{c_0}(c_1).$$

Proof. Immediate from the proof of Lemma B.3.3. □

B.3.2 Basic Properties of the Time-Elapse Function

We next prove lemmas for the time-elapse function in Definition 5.2.10. We redefine the time-elapse function for the generalized runtime model.

Definition B.3.5 (time-elapse function)**.** For each $a \in [0, \infty]$, natural number K and $k \in \{1, \ldots, K\}$, $\mathrm{El}_a^{K,k} : [0, \infty]^K \to [0, \infty]$ is a function defined by

$$\mathrm{El}_a^{K,k}(x_1, \ldots, x_K) = a^k + \sum_{j=1}^{k} \binom{k}{j} a^{k-j} x_j$$

Lemma B.3.6. $\mathrm{El}_a^{K,k}$ is ω-continuous.

Proof. Immediate from (B.3.1) and (B.3.1). □

Lemma B.3.7 (commutativity of $\int$ and $\mathrm{El}_a^{K,k}$)**.** For any probability measure μ on X, any measurable functions $f_1, \ldots, f_n : X \to [0, \infty]$ and $a \in [0, \infty]$, $\mathrm{El}_a^{k,n}$ and integrals commute. That is

$$\int \mathrm{El}_a^{K,k}(f_1(x), \ldots, f_n(x)) d\mu(x) = \mathrm{El}_a^{K,k}\left(\int f_1(x) d\mu(x), \ldots, \int f_n(x) d\mu(x)\right).$$

Proof. By the linearity of integration. □

B.3.3 Characterizing Higher Moments as a Least Fixed Point

We prove Theorem 5.2.7, Theorem 5.2.9 and Theorem 5.2.13 in the generalized runtime model. We first extend Definition 5.2.11 so that an arbitrary reward is allowed.

Definition B.3.8. Let I be an invariant and $C \subseteq I$ be a Borel set. Let $F_K : (S \to [0, \infty]^K) \to (S \to [0, \infty]^K)$ be a function defined by $F_K(c) = (F_{K,1}(c), \ldots, F_{K,K}(c))$ where the k-th component $F_{K,k} : (S \to [0, \infty]^K) \to (S \to [0, \infty])$ of F_K is defined by

$$F_{K,k}(\eta)(c) = \begin{cases} (\overline{\mathbb{X}}(\mathrm{El}_{\mathrm{Rew}(c)}^{K,k} \circ \eta))(c) & c \in I \setminus C \\ 0 & \text{otherwise.} \end{cases}$$

Lemma B.3.9. F_K is ω-continuous.

Proof. Immediate from Lemma B.3.2 and Lemma B.3.6. □

The following theorems generalizes Theorem 5.2.7,5.2.13 and Theorem 5.2.9, respectively.

Theorem B.3.10.

$$\mu F_K \geq \left\langle \overline{\mathbb{M}}_C^{\mathrm{Rew},1}, \ldots, \overline{\mathbb{M}}_C^{\mathrm{Rew},K} \right\rangle$$

for any $c_0 \in I$.

Theorem B.3.11. If $L_N = \emptyset$ and $L_{AN} = \emptyset$,

$$\mu F_K = \left\langle \overline{\mathbb{M}}_C^{\mathrm{Rew},1}, \ldots, \overline{\mathbb{M}}_C^{\mathrm{Rew},K} \right\rangle$$

for any $c_0 \in I$.

Here a function $\langle f_1, \ldots, f_n \rangle$ is defined by $\langle f_1, \ldots, f_K \rangle(x) = (f_1(x), \ldots, f_K(x))$.

To prove Theorem B.3.10 and Theorem B.3.11, we consider an approximation of k-th moments of accumulated rewards up to finite steps.

Definition B.3.12 (accumulated reward up to n steps)**.** Let $\mathrm{Rew}^{c_0}_{C,n} : S^n \to [0, \infty]$ be a measurable function defined by

$$\mathrm{Rew}^{c_0}_{C,n}(c_1 \ldots c_n) = \begin{cases} \sum_{j=0}^{N-1} \mathrm{Rew}(c_j) & \exists N \geq 0.\ c_i \in C \wedge (0 \leq j < N \implies c_j \notin C) \\ \sum_{j=0}^{n-1} \mathrm{Rew}(c_j) & \text{otherwise} \end{cases}$$

The definition of $\mathrm{Rew}^{c_0}_{C,n}$ is similar to $\mathrm{Rew}^{c_0}_{C}$ except that the sum of the value of reward function is restricted to the first n configurations. The next lemma shows a connection between $\mathrm{Rew}^{c_0}_{C}$ and $\mathrm{Rew}^{c_0}_{C,n}$.

Lemma B.3.13. $\{\mathrm{Rew}^{c_0}_{C,n} \circ S^{n \subseteq \omega} : S^\omega \to [0, \infty]\}_n$ is an increasing sequence of functions and its limit is $\mathrm{Rew}^{c_0}_{C}$.

Proof. Given $\rho = c_1 c_2 \cdots \in S^\omega$, there are two cases.

- Assume there exists $N \in \omega$ such that $c_N \in C$ and $0 \leq j < N \implies c_j \notin C$.

$$\mathrm{Rew}^{c_0}_{C,n} \circ S^{n \subseteq \omega}(c_1 c_2 \ldots) = \begin{cases} \displaystyle\sum_{j=0}^{n-1} \mathrm{Rew}(c_j) & \text{if } n < N-1 \\ \displaystyle\sum_{j=0}^{N-1} \mathrm{Rew}(c_j) & \text{if } N-1 \leq n \end{cases}$$

$$\mathrm{Rew}^{c_0}_{C}(c_1 c_2 \ldots) = \sum_{j=0}^{N-1} \mathrm{Rew}(c_j)$$

- Assume $\rho \in (S \setminus C)^\omega$.

$$\mathrm{Rew}^{c_0}_{C,n} \circ S^{n \subseteq \omega}(c_1 c_2 \ldots) = \sum_{j=0}^{n-1} \mathrm{Rew}(c_j)$$

$$\mathrm{Rew}^{c_0}_{C}(c_1 c_2 \ldots) = \sum_{j=0}^{\infty} \mathrm{Rew}(c_j)$$

In both cases, it is easy to prove $\mathrm{Rew}^{c_0}_{C,n} \circ S^{n \subseteq \omega} \leq \mathrm{Rew}^{c_0}_{C,n+1} \circ S^{n+1 \subseteq \omega}$ for each n, and $\mathrm{Rew}^{c_0}_{C} = \sup_{n \in \omega} \left(\mathrm{Rew}^{c_0}_{C,n} \circ S^{n \subseteq \omega}\right)$. □

The k-th moment of $\mathrm{Rew}^{c_0}_{C,n}$ is denoted by $\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n}(c_0)$.

Definition B.3.14 (k-th moment up to n steps)**.** A function $\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n} : S \to [0, \infty]$ is defined as follows.

$$\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n}(c_0) = \int (\mathrm{Rew}^{c_0}_{C,n})^k d\nu^{\sigma}_{c_0,n}$$

A connection between $\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n}$ and $\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma}$ is as follows.

Lemma B.3.15. A sequence $\{\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n}\}_{n\in\omega}$ is increasing and its limit is $\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma}$:

$$\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma} = \sup_{n\in\omega} \mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n}.$$

Proof. The former part is immediate by Lemma B.3.13. The latter part is proved by the following calculation.

$$\begin{aligned}
&\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma}(c_0) \\
&= \int (\mathrm{Rew}^{c_0}_{C})^k \,\mathrm{d}\nu^{\sigma}_{c_0} \\
&= \int \sup_{n\in\omega} (\mathrm{Rew}^{c_0}_{C,n})^k \circ S^{n\subseteq\omega} \,\mathrm{d}\nu^{\sigma}_{c_0} && \text{(by Lemma B.3.13)} \\
&= \sup_{n\in\omega} \int (\mathrm{Rew}^{c_0}_{C,n})^k \circ S^{n\subseteq\omega} \,\mathrm{d}\nu^{\sigma}_{c_0} && \text{(by the monotone convergence theorem)} \\
&= \sup_{n\in\omega} \int (\mathrm{Rew}^{c_0}_{C,n})^k \,\mathrm{d}\big((S^{n\subseteq\omega})_* \nu^{\sigma}_{c_0}\big) && \text{(by Lemma B.1.2)} \\
&= \sup_{n\in\omega} \int (\mathrm{Rew}^{c_0}_{C,n})^k \,\mathrm{d}\nu^{\sigma}_{c_0,n} \\
&= \sup_{n\in\omega} \mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n}(c_0)
\end{aligned}$$

□

Definition B.3.16. For any c and σ, we define a scheduler $\sigma^c = (\sigma^c_t, \sigma^c_a)$ by $\sigma^c_t(\rho) = \sigma_t(c\rho)$ and $\sigma^c_a(\rho) = \sigma_a(c\rho)$.

The following lemma easily follows from the definition of μ^{σ}_{ρ}.

Lemma B.3.17. $\mu^{\sigma^{c_0}}_{\rho} = \mu^{\sigma}_{c_0\rho}$ □

The following lemma expresses the $n+1$ step approximation $\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n+1}$ in terms of the n step approximations $\mathbb{M}^{\mathrm{Rew},1}_{C,\sigma^{c_0},n}, \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma^{c_0},n}$, which plays a crucial role in the induction step in the proof of Theorem B.3.10 and Theorem B.3.11.

Lemma B.3.18. Assume $c_0 \notin C$ and $k \in \{1, \dots, K\}$. For each $n \in \omega$,

$$\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n+1}(c_0) = \mathrm{El}^{K,k}_{\mathrm{Rew}(c_0)} \left(\int_S \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma^{c_0},n}(c_1) \,\mathrm{d}\mu^{\sigma}_{c_0}(c_1), \dots, \int_S \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma^{c_0},n}(c_1) \,\mathrm{d}\mu^{\sigma}_{c_0}(c_1) \right).$$

Proof.

$$\begin{aligned}
&\mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n+1}(c_0) \\
&= \int_{S^{n+1}} (\mathrm{Rew}^{c_0}_{C,n+1})^k \,\mathrm{d}\nu^{\sigma}_{c_0,n+1} \\
&= \int_S \cdots \int_S \Big(\mathrm{Rew}^{c_0}_{C,n+1}(c_1, \dots, c_{n+1})\Big)^k \,\mathrm{d}\mu^{\sigma}_{c_0\dots c_n}(c_{n+1}) \dots \mathrm{d}\mu^{\sigma}_{c_0}(c_1) \\
&= \int_S \cdots \int_S \Big(\mathrm{Rew}(c_0) + \mathrm{Rew}^{c_1}_{C,n}(c_2, \dots, c_{n+1})\Big)^k \,\mathrm{d}\mu^{\sigma}_{c_0\dots c_n}(c_{n+1}) \dots \mathrm{d}\mu^{\sigma}_{c_0}(c_1) \\
&= \int_S \cdots \int_S \left(\sum_{j=0}^{k} \binom{k}{j} (\mathrm{Rew}(c_0))^{k-j} \Big(\mathrm{Rew}^{c_1}_{C,n}(c_2, \dots, c_{n+1})\Big)^j \right) \mathrm{d}\mu^{\sigma}_{c_0\dots c_n}(c_{n+1}) \dots \mathrm{d}\mu^{\sigma}_{c_0}(c_1)
\end{aligned}$$

$$
\begin{aligned}
&= (\mathrm{Rew}(c_0))^k + \sum_{j=1}^{k} \binom{k}{j} (\mathrm{Rew}(c_0))^{k-j} \\
&\qquad \cdot \int_S \left(\int_S \cdots \int_S \left(\mathrm{Rew}^{c_1}_{C,n}(c_2, \dots, c_{n+1}) \right)^j \, \mathrm{d}\mu^{\sigma^{c_0}}_{c_1 \dots c_n}(c_{n+1}) \dots \mathrm{d}\mu^{\sigma^{c_0}}_{c_1}(c_2) \right) \mathrm{d}\mu^{\sigma}_{c_0}(c_1) \\
&= (\mathrm{Rew}(c_0))^k + \sum_{j=1}^{k} \binom{k}{j} (\mathrm{Rew}(c_0))^{k-j} \int_S \int_{S^n} (\mathrm{Rew}^{c_1}_{C,n})^j \, \mathrm{d}\nu^{\sigma^{c_0}}_{c_1,n} \mathrm{d}\mu^{\sigma}_{c_0}(c_1) \\
&= (\mathrm{Rew}(c_0))^k + \sum_{j=1}^{k} \binom{k}{j} (\mathrm{Rew}(c_0))^{k-j} \int_S \mathbb{M}^{\mathrm{Rew},j}_{C,\sigma^{c_0},n}(c_1) \, \mathrm{d}\mu^{\sigma}_{c_0}(c_1)
\end{aligned}
$$

□

Theorem B.3.10.

1. We prove
$$(F_K)^n(\bot) \geq \left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma,n}, \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma,n} \right\rangle$$
for each σ and n by induction on n.

    - If $n = 0$, the l.h.s. and the r.h.s. are equal to 0.
    - If $n > 0$, it suffices to prove that for each c_0, there exists σ' such that
    $$F_K\left(\left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma',n}, \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma',n} \right\rangle\right)(c_0) \geq \left(\mathbb{M}^{\mathrm{Rew},1}_{C,\sigma,n+1}(c_0), \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma,n+1}(c_0) \right)$$
    by the induction hypothesis. If $c_0 \in C$, the l.h.s. and the r.h.s. are equal to 0. If $c_0 \notin C$, we prove
    $$\overline{\mathbb{X}}\left(\mathrm{El}^{K,k}_{\mathrm{Rew}(c_0)} \circ \left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma^{c_0},n}, \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma^{c_0},n} \right\rangle \right)(c_0) \geq \mathbb{M}^{\mathrm{Rew},k}_{C,\sigma,n+1}(c_0).$$
    By Lemma B.3.18, it suffices to prove
    $$\overline{\mathbb{X}}\eta(c_0) \geq \int_S \eta(c_1) \, \mathrm{d}\mu^{\sigma}_{c_0}(c_1)$$
    where
    $$\eta = \mathrm{El}^{K,k}_{\mathrm{Rew}(c_0)} \circ \left(\mathbb{M}^{\mathrm{Rew},1}_{C,\sigma^{c_0},n}, \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma^{c_0},n} \right).$$
    This holds by Lemma B.3.3.

2. We take supremum of (1) with respect to n, and then with respect to σ.
$$\mu F_K \geq \sup_{n \in \omega} \left((F_K)^n(\bot) \right) \geq \left\langle \overline{\mathbb{M}}_C^{\mathrm{Rew},1}, \dots, \overline{\mathbb{M}}_C^{\mathrm{Rew},K} \right\rangle$$

□

Theorem B.3.11. Here, we prove
$$(F_K)^n(\bot) = \left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma,n}, \dots, \mathbb{M}^{\mathrm{Rew},K}_{C,\sigma,n} \right\rangle$$

for each n by induction on n in the same way as Theorem B.3.10 except that we use Lemma B.3.4 instead of Lemma B.3.3.

By the Kleene fixed-point theorem and Lemma B.3.9, we have $\sup_{n\in\omega}\big((F_K)^n(\bot)\big) = \mu F_K$.

$$\begin{aligned}\mu F_K &= \sup_{n\in\omega}\big((F_K)^n(\bot)\big)\\ &= \sup_{n\in\omega}\left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma,n},\dots,\mathbb{M}^{\mathrm{Rew},K}_{C,\sigma,n}\right\rangle\\ &= \left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma},\dots,\mathbb{M}^{\mathrm{Rew},K}_{C,\sigma}\right\rangle\end{aligned}$$

Since there is only one scheduler if $L_N = L_{AN} = \emptyset$, we conclude

$$\mu F_K = \left\langle \mathbb{M}^{\mathrm{Rew},1}_{C,\sigma},\dots,\mathbb{M}^{\mathrm{Rew},K}_{C,\sigma}\right\rangle = \left\langle \overline{\mathbb{M}}^{\mathrm{Rew},1}_{C},\dots,\overline{\mathbb{M}}^{\mathrm{Rew},K}_{C}\right\rangle.$$

□

B.3.4 Ranking Supermartingale for K-th Moments

The following definition and theorem generalize Definition 5.2.12 and Theorem 5.2.13, respectively.

Definition B.3.19 (ranking supermartingale for K-th moments of accumulated rewards)**.** A ranking supermartingale for K-th moments is a function $\eta : S \to \mathbb{R}^K$ such that for each k,

- $\eta_k(c) \geq (\overline{\mathbb{X}}(\mathrm{El}^{K,k}_{\mathrm{Rew}(c)} \circ \eta_k))(c)$ for each $c \in I \setminus C$
- $\eta_k(c) \geq 0$ for each $c \in I$

where $\eta_k : S \to \mathbb{R}$ is defined by $(\eta_1(c),\dots,\eta_K(c)) = \eta(c)$ for each $c \in S$.

Theorem B.3.20. If η is a supermartingale for K-th moments, then for any $c \in I$, $(\overline{\mathbb{M}}^{\mathrm{Rew},1}_{C}(c),\dots,\overline{\mathbb{M}}^{\mathrm{Rew},K}_{C}(c)) \leq \eta(c)$.

Proof. Let $\eta' : S \to [0,\infty]^K$ be a function defined by

$$\eta'(c) = \begin{cases}\eta(c) & c \in I\\ 0 & \text{otherwise.}\end{cases}$$

By Lemma B.3.1, $F_K(\eta') \leq \eta'$ is easily proved. By the Knaster-Tarski theorem, we have $\mu F_K \leq \eta'$. Therefore

$$(\overline{\mathbb{M}}^{\mathrm{Rew},1}_{C}(c),\dots,\overline{\mathbb{M}}^{\mathrm{Rew},K}_{C}(c)) \leq \mu F_K(c) \leq \eta'(c) = \eta(c)$$

for each $c \in I$. □

B.4 Test Programs

We have augmented the standard syntax of randomized program (see e.g. [29]) so that we can specify an invariant and a terminal configuration. To specify an invariant, we can use either of the following syntax.

- $\$\ldots$ specifies an invariant globally.
- $\{\ldots\}$ specifies an invariant locally.

We can specify a terminal configuration by using refute(...).

Listing B.1: (1-1) coupon_collector

```
$ 0 <= c0 and c0 <= 1 and 0 <= c1 and c1 <= 1

c0 := 0;
c1 := 0;

while true do
  if prob(0.5) then
    c0 := 1
  else
    c1 := 1
  fi;
  refute (c0 + c1 > 1)
od
```

Listing B.2: (1-2) coupon_collector4

```
$ 0 <= c0 and c0 <= 1 and 0 <= c1 and c1 <= 1 and 0 <= c2 and c2 <= 1 and 0
     <= c3 and c3 <= 1

c0 := 0;
c1 := 0;
c2 := 0;
c3 := 0;

while true do
  if prob(0.5) then
    if prob(0.5) then
      c0 := 1
    else
      c1 := 1
    fi
  else
    if prob(0.5) then
      c2 := 1
    else
      c3 := 1
    fi
  fi;
  refute (c0 + c1 + c2 + c3 > 3)
od
```

Listing B.3: (2-1) random_walk_1d_intvalued

```
{ true } x := 1;

{x >= 1} while true do
{x >= 1}   if prob(0.6) then
{x >= 1}     x := x - 1
           else
{x >= 1}     x := x + 1
           fi;
{x >= 0}   refute (x < 1)
         od
```

Listing B.4: (2-2) random_walk_1d_realvalued

```
{ true }                          x := 2;

{ x >= 0 }                        while true do
{ x >= 0 }                          if prob(0.7) then
{ x >= 0 }                            z := Unif(0,1);
{ x >= 0 and 0 <= z and z <= 1 }      x := x - z
                                    else
{ x >= 0 }                            z := Unif(0,1);
{ x >= 0 and 0 <= z and z <= 1 }      x := x + z
                                    fi;
{ x >= -1 }                         refute (x < 0)
                                  od
```

Listing B.5: (2-3) random_walk_1d_adversary

```
{ true }               x := 2;

{ 0 <= x and x <= 13 } while true do
{ 0 <= x and x <= 10 }   if prob(0.8) then
{ 0 <= x and x <= 10 }     skip
                         else
{ 0 <= x and x <= 10 }     if prob(0.5) then
{ 0 <= x and x <= 10 }       x := x + 1
                           else
{ 0 <= x and x <= 10 }       x := x + 2
                           fi
                         fi;
{ 0 <= x and x <= 12 }   if * then
{ 0 <= x and x <= 12 }     if prob(0.875) then
{ 0 <= x and x <= 12 }       x := x - 1
                           else
{ 0 <= x and x <= 12 }       skip
                           fi
                         else
{ 0 <= x and x <= 12 }     if prob(0.8) then
{ 0 <= x and x <= 12 }       skip
                           else
{ 0 <= x and x <= 12 }       if prob(0.5) then
{ 0 <= x and x <= 12 }         x := x + 1
                             else
{ 0 <= x and x <= 12 }         x := x + 2
                             fi
                           fi;
{ 0 <= x and x <= 14 }     x := x - 1
                         fi;
{ 0 <= x and x <= 13 }   refute (x <= 0)
                       od
```

Listing B.6: (2-4) random_walk_2d_demonic

```
{ true }                                      x := 2;
{ x = 2 }                                     y := 2;
{ 0 <= x and 0 <= y }                         while true do
{ 0 <= x and 0 <= y }                           if * then
{ 0 <= x and 0 <= y }                             z := Unif (-2,1);
{ 0 <= x and 0 <= y and -2 <= z and z <= 1 }      x := x + z
                                                else
{ 0 <= x and 0 <= y }                             z := Unif (-2,1);
{ 0 <= x and 0 <= y and -2 <= z and z <= 1 }      y := y + z
                                                fi;
{ -2 <= x and -2 <= y }                         refute (x <= 0);
{ 0 <= x and -2 <= y }                          refute (y <= 0)
                                              od
```

Listing B.7: (2-5) random_walk_2d_variant

```
{ true }                           x := 3;
{ x = 3 }                          y := 2;
{ x >= y }                         while true do
{ x >= y }                           if * then
{ x >= y }                             if prob(0.7) then
{ x >= y }                               z := Unif (-2,1);
{ x >= y and -2 <= z and z <= 1 }        x := x + z
                                       else
{ x >= y }                               z := Unif (-2,1);
{ x >= y and -2 <= z and z <= 1 }        y := y + z
                                       fi
                                     else
{ x >= y }                             if prob(0.7) then
{ x >= y }                               z := Unif (-1,2);
{ x >= y and -1 <= z and z <= 2 }        y := y + z
                                       else
{ x >= y }                               z := Unif (-1,2);
{ x >= y and -1 <= z and z <= 2 }        x := x + z
                                       fi
                                     fi;
{ x >= y + 2 }                       refute (x <= y)
                                   od
```

B.5 Detailed Comparison with Existing Work

B.5.1 Comparison with [26]

In the literature on martingale-based methods, the one closest to this work is [26]. Among its contribution is the analysis of tail probabilities by either of the following two combinations:

- *difference-bounded* ranking supermartingales and the corresponding choice of concentration inequality (namely Azuma's martingale concentration lemma); and
- (not necessarily difference-bounded) ranking supermartingales and Markov's concentration inequality.

While implementation and experiments are lacking in [26], we can make the following theoretical comparison between these two methods and ours.

- The first method (with difference-bounded supermartingales) requires trying many difference bounds c, synthesizing a martingale for each c, and picking the best one. This "try many and pick the best" workflow is much like in [29]; it increases the computational cost, especially in the case a polynomial template is used (where a single synthesis procedure takes tens of seconds).
- The second method corresponds precisely to the special case of our method where we restrict to the first moment. We argued that using higher moments is crucial in obtaining tighter bounds as the deadline becomes large, theoretically (Section 5.3) and experimentally (Section 5.5).

B.5.2 Comparison with [59]

In the predicate-transformer approach, the work [59] is the closest to ours, in that it studies *variance* of runtimes of randomized programs. The main differences are as follows: 1) computing tail probabilities is not pursued; 2) their extension from mean to variance involves an additional variable τ, which poses a challenge in automated synthesis as well as in generalization to even higher moments; and 3) they do not pursue automated analysis.

Let us elaborate on the above point 2). In syntax-based static approaches to estimating variances or second moments, it is inevitable to simultaneously reason about both first and second moments. See Def. 5.2.3. In this work, we do so systematically by extending a notion of supermartingale into a *vector-valued* notion; this way our theory generalizes to moments higher than the second in a straight-forward manner. In contrast, in [59], an additional variable τ—which stands for the elapsed time—is used for mixing first and second moments.

Besides the problem of generalizing to higher moments, a main drawback of this approach in [59] is that the degrees of templates become bigger when it comes to automated synthesis. For example, due to the use of τ^2 in the condition for $\hat{X}$ in [59, Thm. 7], if the template for τ is of degree k, the template for $\hat{X}$ is necessarily of degree $2k$ or higher. One consequence is that a fully LP-based implementation of the approach of [59] becomes hard, while it is possible in the current work (see Section 5.5).

Let us also note that the work [59] focuses on precondition calculi and does not discuss automated synthesis or analysis.

B.6 An Example of Polynomially Decreasing Tail Probability

We show that there exists a randomized program such that the tail probability of the runtime is polynomially decreasing (not exponentially decreasing). A similar example can be found in [26, Example 8].

```
$ 0 <= r and r <= 1 and 0 <= n
n := 1;
r := Unif(0, 1);
while r * (n + 1) * (n + 1) <= n * n do
    r := Unif(0, 1);
    n := n + 1
od
```

Let T_l be a random variable that represents the number of iterations. As the program executes the loop body with probability $\frac{n^2}{(n+1)^2}$ in the n-th iteration, the tail probability of the runtime of the program is polynomially decreasing:

$$\Pr(T_l \geq d) = \left(\frac{1}{2}\right)^2 \cdots \left(\frac{d}{d+1}\right)^2 = \left(\frac{1}{d+1}\right)^2.$$

We can apply the polynomial template-based algorithm for this program (but cannot apply the linear one since the condition in the while statement is not linear). Our implementation gives the following upper bound of the first moment of the runtime. This upper bound can be used to bound tail probabilities, via the inequality in Prop. 5.3.2.

moment	upper bound	time (sec)	degree
1st	13.15	534.575	3

Bibliography

[1] llvm2KITTeL. `https://github.com/hkhlaaf/llvm2kittel`.

[2] StarExec. `https://www.starexec.org`.

[3] T2 temporal logic prover. `https://github.com/mmjb/T2`.

[4] Termination Competition 2020: C Integer. `https://termcomp.github.io/Y2020/job_41519.html`.

[5] Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: an efficient approach to termination of probabilistic programs. *PACMPL*, 2(POPL):34:1–34:32, 2018.

[6] Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, Shin-ya Katsumata, and Tetsuya Sato. Higher-order probabilistic adversarial computations: Categorical semantics and program logics. *Proceedings of the ACM on Programming Languages*, 5(ICFP):1–30, August 2021.

[7] Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. A relational logic for higher-order programs. *Proceedings of the ACM on Programming Languages*, 1(ICFP):1–29, August 2017.

[8] Alejandro Aguirre and Shin-ya Katsumata. 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.

[9] Danel Ahman. *Fibred Computational Effects*. PhD Thesis, University of Edinburgh, 2017.

[10] Danel Ahman. Handling fibred algebraic effects. *Proceedings of the ACM on Programming Languages*, 2(POPL):1–29, January 2018.

[11] Danel Ahman, Neil Ghani, and Gordon D. Plotkin. Dependent types and fibred computational effects. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures*, volume 9634, pages 36–54. Springer Berlin Heidelberg, 2016.

[12] Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *SAS '10*, pages 117–133. Springer, 2010.

[13] Robert B. Ash and Catherine A. Doleans-Dade. *Probability and Measure Theory.* Academic Press, second edition, 1999.

[14] Gilles Barthe, Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Pierre-Yves Strub. 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*, pages 55–68, Mumbai, India, 2015. ACM Press.

[15] Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. How long, O bayesian network, will I sample thee? - A program analysis perspective on expected sampling times. In *ESOP*, volume 10801 of *Lecture Notes in Computer Science*, pages 186–213. Springer, 2018.

[16] Andrej Bauer and Matija Pretnar. Programming with algebraic effects and handlers. *Journal of Logical and Algebraic Methods in Programming*, 84(1):108–123, January 2015.

[17] Amir M. Ben-Amram, Jesús J. Doménech, and Samir Genaim. Multiphase-linear ranking functions and their relation to recurrent sets. In *SAS '19*, pages 459–480. Springer, 2019.

[18] Amir M. Ben-Amram and Samir Genaim. Ranking functions for linear-constraint loops. *Journal of the ACM*, 61(4), July 2014.

[19] Amir M. Ben-Amram and Samir Genaim. On multiphase-linear ranking functions. In *CAV '17*, pages 601–620. Springer, 2017.

[20] Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan, and Andrey Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II: Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, volume 9300 of *LNCS*, pages 24–51. Springer, 2015.

[21] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration Inequalities: A Nonasymptotic Theory of Independence.* Oxford University Press, 2013.

[22] Marc Brockschmidt, Thomas Ströder, Carsten Otto, and Jürgen Giesl. Automated detection of non-termination and NullPointerExceptions for Java bytecode. In *FoVeOOS '11*, volume 7421 of *LNCS*, pages 123–141. Springer, 2012.

[23] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 511–526. Springer, 2013.

[24] Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic Program Analysis with Martingales. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell,

Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Natasha Sharygina, and Helmut Veith, editors, *Computer Aided Verification*, volume 8044, pages 511–526. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[25] Adrien Champion, Tomoya Chiba, Naoki Kobayashi, and Ryosuke Sato. ICE-based refinement type discovery for higher-order functional programs. In *TACAS '18*, volume 10805 of *LNCS*, pages 365–384. Springer, 2018.

[26] Krishnendu Chatterjee and Hongfei Fu. Termination of nondeterministic recursive probabilistic programs. *CoRR*, abs/1701.02944, 2017.

[27] Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through positivstellensatz's. In *CAV (1)*, volume 9779 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2016.

[28] Krishnendu Chatterjee, Hongfei Fu, Petr Novotný, and Rouzbeh Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. *ACM Trans. Program. Lang. Syst.*, 40(2):7:1–7:45, 2018.

[29] Krishnendu Chatterjee, Petr Novotný, and Dorde Zikelic. Stochastic invariants for probabilistic termination. In *POPL*, pages 145–160. ACM, 2017.

[30] Tugrul Dayar and Nail Akar. Computing moments of first passage times to a subset of states in markov chains. *SIAM J. Matrix Analysis Applications*, 27(2):396–412, 2005.

[31] Paulo Emílio de Vilhena and François Pottier. A separation logic for effect handlers. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–28, January 2021.

[32] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, August 1975.

[33] Benjamin Doerr. Probabilistic tools for the analysis of randomized optimization heuristics. *CoRR*, abs/1801.06733, 2018.

[34] Stephen Dolan, Spiros Eliopoulos, Daniel Hillerström, Anil Madhavapeddy, K. C. Sivaramakrishnan, and Leo White. Concurrent system programming with effect handlers. In Meng Wang and Scott Owens, editors, *Trends in Functional Programming*, volume 10788, pages 98–117. Springer International Publishing, Cham, 2018.

[35] Fabian Emmes, Tim Enger, and Jürgen Giesl. Proving non-looping non-termination automatically. In *IJCAR '12*, volume 7364 of *LNCS*, pages 225–240. Springer, 2012.

[36] P. Ezudheen, Daniel Neider, Deepak D'Souza, Pranav Garg, and P. Madhusudan. Horn-ICE learning for synthesizing invariants and contracts. *Proceedings of the ACM on Programming Languages*, 2(OOPSLA):131:1–131:25, October 2018.

[37] Grigory Fedyukovich, Yueling Zhang, and Aarti Gupta. Syntax-guided termination analysis. In *CAV '18*, volume 10981 of *LNCS*, pages 124–143. Springer, 2018.

[38] Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In *POPL*, pages 489–501. ACM, 2015.

[39] Cormac Flanagan. Hybrid type checking. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL'06*, pages 245–256, Charleston, South Carolina, USA, 2006. ACM Press.

[40] Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

[41] Tim Freeman and Frank Pfenning. Refinement types for ML. *ACM SIGPLAN Notices*, 26(6):268–277, June 1991.

[42] Soichiro Fujii, Shin-ya Katsumata, and Paul-André Melliès. Towards a Formal Theory of Graded Monads. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures*, volume 9634, pages 513–530. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[43] Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. ICE: A robust framework for learning invariants. In *CAV '14*, pages 69–87. Springer, 2014.

[44] Pranav Garg, Daniel Neider, P. Madhusudan, and Dan Roth. Learning invariants using decision trees and implication counterexamples. In *POPL '16*, pages 499–512. ACM, 2016.

[45] The GNU linear programming kit. `https://www.gnu.org/software/glpk/`.

[46] Laure Gonnord, David Monniaux, and Gabriel Radanne. Synthesis of ranking functions using extremal counterexamples. In *PLDI '15*, pages 608–618. ACM, 2015.

[47] Ashutosh Gupta, Thomas A. Henzinger, Rupak Majumdar, Andrey Rybalchenko, and Ru-Gang Xu. Proving non-termination. In *POPL '08*, pages 147–158. ACM, 2008.

[48] Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Termination analysis by learning terminating programs. In *CAV '14*, pages 797–813. Springer, 2014.

[49] Claudio Hermida. *Fibrations, logical predicates and indeterminates.* PhD Thesis, University of Edinburgh, UK, 1993.

[50] Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, Reykjavik, Iceland, June 2017. IEEE.

[51] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, October 1969.

[52] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, New York, NY, USA, 2nd edition, 2012.

[53] Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier, paperback edition, 2001.

[54] Pushpak Jagtap, Sadegh Soudjani, and Majid Zamani. Temporal logic verification of stochastic systems using barrier certificates. In Lahiri and Wang [74], pages 177–193.

[55] Christian Jansson. Termination and verification for ill-posed semidefinite programming problems. *Optimization Online*, 2005.

[56] Christian Jansson. Vsdp: A matlab software package for verified semidefinite programming. *NOLTA*, pages 327–330, 2006.

[57] Christian Jansson, Denis Chaykin, and Christian Keil. Rigorous error bounds for the optimal value in semidefinite programming. *SIAM J. Numerical Analysis*, 46(1):180–200, 2007.

[58] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *Journal of Functional Programming*, 28:e20, 2018.

[59] Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Inferring covariances for probabilistic programs. In *QEST*, volume 9826 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2016.

[60] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected run-times of probabilistic programs. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 364–389. Springer, 2016.

[61] Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM*, 65(5):30:1–30:68, 2018.

[62] Joost-Pieter Katoen, Annabelle McIver, Larissa Meinicke, and Carroll C. Morgan. Linear-invariant generation for probabilistic programs: - automated support for proof-based methods. In *Static Analysis - 17th International Symposium, SAS 2010*, pages 390–406, 2010.

[63] Shin-ya Katsumata. A semantic formulation of $\top\top$-lifting and logical predicates for computational metalanguage. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, and Luke Ong, editors, *Computer Science Logic*, volume 3634, pages 87–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[64] Shin-ya Katsumata. Relating computational effects by $\top\top$-lifting. *Information and Computation*, 222:228–246, January 2013.

[65] Shin-ya Katsumata. Parametric effect monads and semantics of effect systems. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '14*, pages 633–645, San Diego, California, USA, 2014. ACM Press.

[66] Shin-ya Katsumata. private communication, 2020.

[67] Kenneth Knowles and Cormac Flanagan. Compositional reasoning and decidable checking for dependent contract types. In *Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification - PLPV '09*, page 27, Savannah, GA, USA, 2008. ACM Press.

[68] Dexter Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981.

[69] Siddharth Krishna, Christian Puhrsch, and Thomas Wies. Learning invariants using decision trees. *CoRR*, abs/1501.04725, 2015.

[70] Satoshi Kura. A General Semantic Construction of Dependent Refinement Type Systems, Categorically. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures*, volume 12650, pages 406–426. Springer International Publishing, Cham, 2021.

[71] Satoshi Kura, Hiroshi Unno, and Ichiro Hasuo. Decision Tree Learning in CEGIS-Based Termination Analysis. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification*, volume 12760, pages 75–98. Springer International Publishing, Cham, 2021.

[72] Satoshi Kura, Natsuki Urabe, and Ichiro Hasuo. Tail Probabilities for Randomized Program Runtimes via Martingales for Higher Moments. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*, volume 11428 of *Lecture Notes in Computer Science*, pages 135–153. Springer, 2019.

[73] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011.

[74] Shuvendu K. Lahiri and Chao Wang, editors. *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*. Springer, 2018.

[75] Joachim Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge [Cambridgeshire] ; New York, 1986.

[76] Nico Lehmann and Éric Tanter. Gradual refinement types. *ACM SIGPLAN Notices*, 52(1):775–788, May 2017.

[77] Daan Leijen. Koka: Programming with row polymorphic effect types. *Electronic Proceedings in Theoretical Computer Science*, 153:100–126, June 2014.

[78] Jan Leike and Matthias Heizmann. Ranking templates for linear loops. In *TACAS '14*, volume 8413 of *LNCS*, pages 172–186. Springer, 2014.

[79] Paul Blain Levy. *Call-by-push-value*. PhD Thesis, Queen Mary University of London, UK, 2001.

[80] Žiga Lukšič and Matija Pretnar. Local algebraic effect theories. *Journal of Functional Programming*, 30:e13, 2020.

[81] Kenji Maillard, Danel Ahman, Robert Atkey, Guido Martínez, Cătălin Hriţcu, Exequiel Rivas, and Éric Tanter. Dijkstra monads for all. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, July 2019.

[82] Dylan McDermott and Alan Mycroft. Extended Call-by-Push-Value: Reasoning About Effectful Programs and Evaluation Order. In Luís Caires, editor, *Programming Languages and Systems*, volume 11423, pages 235–262. Springer International Publishing, Cham, 2019.

[83] Paul-André Melliès and Noam Zeilberger. Functors are Type Refinement Systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '15*, pages 3–16, Mumbai, India, 2015. ACM Press.

[84] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, July 1991.

[85] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Trans. Program. Lang. Syst.*, 18(3):325–353, 1996.

[86] Xinming Ou, Gang Tan, Yitzhak Mandelbaum, and David Walker. Dynamic Typing with Dependent Types. In Jean-Jacques Levy, Ernst W. Mayr, and John C. Mitchell, editors, *Exploring New Frontiers of Theoretical Informatics*, volume 155, pages 437–450. Kluwer Academic Publishers, Boston, 2004.

[87] Saswat Padhi, Todd D. Millstein, Aditya V. Nori, and Rahul Sharma. Overfitting in synthesis: Theory and practice. In *CAV '19*, volume 11561 of *LNCS*, pages 315–334. Springer, 2019.

[88] Saswat Padhi, Rahul Sharma, and Todd D. Millstein. Data-driven precondition inference with learned features. In *PLDI '16*, pages 42–56, 2016.

[89] Gordon Plotkin and John Power. Adequacy for algebraic effects. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Furio Honsell, and Marino Miculan, editors, *Foundations of Software Science and Computation Structures*, volume 2030, pages 1–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[90] Gordon Plotkin and John Power. Notions of computation determine monads. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Mogens Nielsen, and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures*, volume 2303, pages 342–356. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[91] Gordon Plotkin and John Power. Algebraic Operations and Generic Effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[92] Gordon Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4):23, December 2013.

[93] Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI '04*, volume 2937 of *LNCS*, pages 239–251. Springer, 2004.

[94] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. Liquid types. In *Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '08*, page 159, Tucson, AZ, USA, 2008. ACM Press.

[95] Pierre Roux, Mohamed Iguernlala, and Sylvain Conchon. A non-linear arithmetic procedure for control-command software verification. In *TACAS (2)*, volume 10806 of *Lecture Notes in Computer Science*, pages 132–151. Springer, 2018.

[96] Pierre Roux, Yuen-Lam Voronin, and Sriram Sankaranarayanan. Validating numerical semidefinite programming solvers for polynomial invariants. *Formal Methods in System Design*, 53(2):286–312, 2018.

[97] J. Rushby, S. Owre, and N. Shankar. Subtypes for specifications: Predicate subtyping in PVS. *IEEE Transactions on Software Engineering*, 24(9):709–720, Sept./1998.

[98] Yuki Satake, Hiroshi Unno, and Hinata Yanagi. Probabilistic inference for predicate constraint satisfaction. In *Proceedings of AAAI 2020*, 2020.

[99] Tetsuya Sato. Approximate relational hoare logic for continuous random samplings. *Electronic Notes in Theoretical Computer Science*, 325:277–298, October 2016.

[100] Tetsuya Sato, Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Justin Hsu. Formal verification of higher-order probabilistic programs: Reasoning about approximation, convergence, Bayesian inference, and optimization. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, January 2019.

[101] Tetsuya Sato, Gilles Barthe, Marco Gaboardi, Justin Hsu, and Shin-ya Katsumata. Approximate Span Liftings: Compositional Semantics for Relaxations of Differential Privacy. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14, Vancouver, BC, Canada, June 2019. IEEE.

[102] Konrad Schmüdgen. Thek-moment problem for compact semi-algebraic sets. *Mathematische Annalen*, 289(1):203–206, Mar 1991.

[103] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1986.

[104] Dana Scott. Relating theories of the lambda calculus. *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 403–450, 1980.

[105] SDPT3. `http://www.math.nus.edu.sg/~mattohkc/SDPT3.html`.

[106] A. Simpson and G. Plotkin. Complete axioms for categorical fixed-point operators. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332)*, pages 30–41, Santa Barbara, CA, USA, 2000. IEEE Comput. Soc.

[107] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *ASPLOS XII*, pages 404–415. ACM, 2006.

[108] SOSTOOLS. `http://sysos.eng.ox.ac.uk/sostools/`.

[109] Jacob Steinhardt and Russ Tedrake. Finite-time regional verification of stochastic non-linear systems. *I. J. Robotics Res.*, 31(7):901–923, 2012.

[110] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. *Journal of Functional Programming*, 23(4):402–451, July 2013.

[111] Nikhil Swamy, Joel Weinberger, Cole Schlesinger, Juan Chen, and Benjamin Livshits. Verifying higher-order programs with the dijkstra monad. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '13*, page 387, Seattle, Washington, USA, 2013. ACM Press.

[112] Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. Ranking and repulsing supermartingales for reachability in probabilistic programs. In Lahiri and Wang [74], pages 476–493.

[113] Terence Tao. *An Introduction to Measure Theory.* American Mathematical Society, 2011.

[114] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5, 06 1955.

[115] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank D. Wood. Design and implementation of probabilistic programming language anglican. In *IFL*, pages 6:1–6:12. ACM, 2016.

[116] Hiroshi Unno, Yuki Satake, and Tachio Terauchi. Relatively complete refinement type system for verification of higher-order non-deterministic programs. *Proceedings of the ACM on Programming Languages*, 2:1–29, January 2018.

[117] Hiroshi Unno, Yuki Satake, Tachio Terauchi, and Eric Koskinen. Program verification via predicate constraint satisfiability modulo theories. *CoRR*, abs/2007.03656, 2020.

[118] Hiroshi Unno, Tachio Terauchi, and Eric Koskinen. Constraint-based relational verification. In *CAV '21*. Springer, 2021.

[119] Natsuki Urabe, Masaki Hara, and Ichiro Hasuo. Categorical liveness checking by corecursive algebras. In *Proc. of LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[120] Caterina Urban. The abstract domain of segmented ranking functions. In *SAS '13*, volume 7935 of *LNCS*, pages 43–62. Springer, 2013.

[121] Caterina Urban, Arie Gurfinkel, and Temesghen Kahsai. Synthesizing ranking functions from bits and pieces. In *TACAS '16*, page 54–70. Springer, 2016.

[122] Caterina Urban and Antoine Miné. An abstract domain to infer ordinal-valued ranking functions. In *ESOP '14*, pages 412–431. Springer, 2014.

[123] Caterina Urban and Antoine Miné. A decision tree abstract domain for proving conditional termination. In *SAS '14*, pages 302–318. Springer, 2014.

[124] Niki Vazou, Patrick M. Rondon, and Ranjit Jhala. Abstract Refinement Types. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Matthias Felleisen, and Philippa Gardner, editors, *Programming Languages and Systems*, volume 7792, pages 209–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[125] Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. Refinement types for Haskell. In *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming - ICFP '14*, pages 269–282, Gothenburg, Sweden, 2014. ACM Press.

[126] Hongwei Xi and Frank Pfenning. Eliminating array bound checking through dependent types. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation - PLDI '98*, pages 249–257, Montreal, Quebec, Canada, 1998. ACM Press.

[127] He Zhu, Stephen Magill, and Suresh Jagannathan. A data-driven CHC solver. In *PLDI '18*, pages 707–721. ACM, 2018.