



HAL
open science

Formal Methods for Quantum Programming Languages

Dongho Lee

► **To cite this version:**

Dongho Lee. Formal Methods for Quantum Programming Languages. Formal Languages and Automata Theory [cs.FL]. Université Paris-Saclay, 2022. English. NNT: 2022UPASG059 . tel-03895847

HAL Id: tel-03895847

<https://theses.hal.science/tel-03895847v1>

Submitted on 13 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formal Methods for Quantum Programming Languages

Méthodes Formelles pour les Langages de Programmation Quantiques

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, sciences et technologies de l'information et de la
communication (STIC)

Spécialité de doctorat: Mathématiques et Informatique

Graduate School : Informatique et sciences du numérique, Référent :
CentraleSupélec

Thèse préparée dans l'unité de recherche Laboratoire Méthodes Formelles (Université
Paris-Saclay, CNRS, ENS Paris-Saclay), sous la direction de Frédéric BOULANGER,
professeur, le co-encadrement de Benoît VALIRON, professeur, et le co-encadrement de
Valentin PERRELLE, ingénieur de recherche

Thèse soutenue à Paris-Saclay, le 21 juillet 2022, par

Dongho LEE

Composition du jury

Delia Kesner Professeure, IRIF, Université de Paris	Présidente & examinatrice
Ugo Dal Lago Professeur, Université de Bologne	Rapporteur & examinateur
Lionel Vaux Maître de conférences, LdP, Université d'Aix-Marseille	Rapporteur & examinateur
Chantal Keller Maîtresse de conférences, LMF, Université Paris-Saclay	Examinatrice
Frederic Boulanger Professeur, CentraleSupélec, Université Paris-Saclay	Directeur de thèse

Membres invités

Benoît Valiron Professeur, CentraleSupélec, Université Paris-Saclay	Encadrant
Valentin Perrelle Ingénieur de recherche, CEA LIST, Université Paris-Saclay	Encadrant

Acknowledgements

Four years ago, when I started this thesis, I did not know that it was going to be about the categorical semantics of a programming language. Six years ago, when I started my master's degree, I did not imagine that I was going to do research in programming languages. Ten years ago, when I was in college, I knew almost nothing about logic or quantum computation. And for my whole life, before I decided to do a thesis in France, I had never imagined myself speaking in French! If I compare life with a journey, it was not easy for me to arrive here (not only because it was challenging but also because it was not likely to happen), and it is wondrous to me that I have finished my thesis. With this wonder, I would like to thank all people and groups of people who have directly or indirectly, academically or non-academically, influenced and helped me finish this thesis.

Having studied in France is one of the privileges given to me. I received all the support and supervision required for research and thesis writing. I could enjoy and benefit from the country's profound history, diverse culture, delicious food and wines, and different forms of exhibitions and concerts. But, above all, I was privileged to meet awesome people in France. First of all, I would like to thank my supervisors—Benoît Valiron and Valentin Perrelle—who have given me a manifold of help from settling in France to research and academic career. I first thank Benoît for introducing me to the categorical semantics of programming language and guiding me through many discussions to the main results of this thesis. He was like an oracle during my thesis whose words have often been realized and verified later. These prophecies mainly happened when I did not follow him initially and found out what he had been talking about after doing some work. He was also a very considerate and fun leader inside and outside the lab. I thank him for inviting me to Christmas dinners, where I could enjoy his famous dishes like tajine and braised octopus, followed by many desserts! It was comforting, especially when I could not travel a lot to visit home in South Korea during the holidays and the pandemic.

I would also like to thank Valentin, a very generous supervisor throughout the thesis, for giving me help and support whenever I needed it. In particular, I would like to thank him for being patient and allowing me enough time when I took a turn toward the category theory and spent much time studying it. I felt lucky to be able to do that since I could imagine that the pressure of producing results hinders Ph.D. students from studying much time, although it would bring them a better understanding of their research subjects. Besides, I always enjoyed

following him to the typical coffee breaks of the LSL, once in the morning and once in the afternoon at the corner of the second floor of the building at the Nano Innov, where I enjoyed witnessing the French culture of discussion and feel more integrated despite my poor French comprehension.

Secondly, I would like to thank all my colleagues from the CEA and the LRI, which became the LMF later. I thank LSL colleagues who have greeted me warmly since my first day—my officemates: Alexandre Chabot, Guillaume Girol, and Bernard Nongpoh; my fellow Ph.D. students: Dara Ly, Yaëlle Vinçont, Hugo Illous, Maxime Jacquemin, Julien Girard-Satabin, Lesly-Ann Daniel, Olivier Nicole, Virgile Robles, Manh Dung Nguyen, Thibault Martin, Zeinab Nehai, Yackolley Amoussou-Guenou, Charles B Mamidisetti, Myriam Clouet; permanent researchers: Florent Kirchner, Thibaud Antignac, Andre Maroneze, Julien Signoles, Virgile Prevosto, David Bühler, Hichem Zakaria Chihani, François Bobot; and Frédérique Descreaux who helped me solve all the infamous administrative problems. In addition, I would like to send special thanks to Sébastien Bardin and Christophe Chareton for their helpful advice on research and all the exciting discussions on quantum computation and formal methods during numerous coffee breaks.

Likewise, I owe many thanks to my colleagues at the LRI and the LMF. Starting from my officemate, Lulu He has been very friendly and helpful. I have also been grateful for my neighbors in the lab: Frédéric Boulanger, who was an accommodating thesis director with prompt responses and aides each time I needed them; Lina Ye and Safouan Taha, who have greeted me with a smile each time; Zhaowei Xu, who had been a sincere collaborator and advisor with whom I can talk freely about anything and with whom I enjoyed discovering different Chinese restaurants in Paris. I am grateful to all who helped me to settle in France, including Marc Baboulin, who generously presented me to a postdoc from South Korea, and Nicolas Nalpon, who took me to different places in Paris.

The LRI was divided into two parts, and the VALS team, to which I belonged, was merged with the LSV, changing its name to LMF. Although the reallocation of the office did not happen until I finished my thesis, this change allowed me to meet colleagues from both laboratories. I cannot say that I know all the Ph.D. students from the laboratories since they are both quite large, but it was always fun to discuss different things with them on various subjects and learn from them. I am grateful to Han Han, Amrita Suresh, Rébecca Zucchini, and Amélie Ledain for being friendly colleagues and helping me integrate into the lab.

I also want to distinguish and thank my colleagues from the QuaCS team—Kostia Chardonnet, Agustín Pablo Borgna, Renaud Vilmart, Titouan Carette, Vitor Fernandes, Thibault Fredon, Vladimir Zamdzhiev, Marc De Visme, Louis Lemonnier, Nicolas Heurtel, Luidnel Maignan, Pablo Arnault, and Pablo Arrighi. I was always excited about discussing and sharing ideas and experiences on different

subjects with them, where I learned much about research. It was just wonderful to spend time with these remarkable people with whom I share many interests.

In addition, many people helped me outside the laboratory as well. They made me feel not lonely and filled my life with everlasting memories. Without the help of Science Accueil, I could not find housing when I first came to France. I appreciated cultural excursions and French courses—that I did not miss one semester except for the last—which helped me understand France’s people, history, language, and culture. These experiences filled me with curiosity by distinguishing France from anywhere in the world and let me try to learn more by myself through different mediums like news, books, and communication with others. For this reason, I would like to thank Sophie Langrognet, Jean Bertsch, and all other members of Science Accueil and French teachers.

I want to describe my special thanks to Jules Jacobs, whom I met at the Oregon Programming Language Summer School in 2019. He kindly invited me to the Netherlands, where I spent the beautiful Christmas holidays with him and his family in Nuth later in the year. Through enlightening conversations and working on concrete problems, he introduced me to many different subjects in mathematics. Among many questions we discussed, I am particularly indebted to his question on the quantum types in programming languages. When he asked why quantum types are Hilbert spaces and not something else like vectors in Hilbert space, I could not answer but say that the type systems of quantum programming languages like quantum λ -calculus, Quipper, and Qwire are defined as they are. This question has lingered in my mind and led me to new insights in my research. This kind of revelation through freely enquiring and fun and enriching discussions with Jules was pure joy that made me visit him twice more; each time, he welcomed me with pleasure.

If there is a benefit that the isolation induced by the covid brought me, then it would be the online French course with my french teacher, Régis Albert, in South Korea. I took his French courses for two months in South Korea before coming to France. After the covid spread out and I was isolated in my room, I began retaking his classes online, where I could talk about life and questions I had in France while learning French. In addition, I would like to thank all my friends from the Maison de la Corée and the Cité Universitaire and my classmates from the French course by Science Accueil, who allowed me plenty of pleasant moments throughout the thesis. Furthermore, I thank Kwan-djou Djo, who has given me practical help in finding housing in Paris while protecting me from any possible difficulties in France. Finally, I would like to thank Pastor Bruno Aussant and all the members of my Church who provided me with invaluable messages and excellent food each week.

Next, I would like to thank some of those I met before starting my thesis. Some of their influence was so crucial to me that I cannot imagine doing my thesis without it. Starting from those I met at the University of Pennsylvania during my master's degree, Steve Zdancewic, with whom I did an internship, has introduced me to research on programming languages for quantum computers. I was unfamiliar with programming languages or quantum computation when I started my internship. However, while doing projects with his Ph.D. students Robert Rand and Jennifer Paykin, I could make familiar with the proof assistant-COQ-and quantum circuit description language-QWire. Those are exciting subjects that I am still interested in, and, most of all, this experience was essential when I was looking for a Ph.D. thesis position and doing my thesis. For this reason, I am always grateful to all three-Steve, Robert, and Jennifer.

Before I started an internship with Steve, I was fascinated by logic, particularly (finite) model theory, while taking a sequence of courses taught by Scott Weinstein, Stepan Kuznetsov, and Val Tannen. To begin with, Scott's course on logic and computability based on the model theory has added a new dimension to my perspective and breathed life into my study of formal logic. The model theory was so inspiring that I decided to do an independent study under his supervision. The second part of the course was on proof theory taught by Stepan. Another course on friendly logic, taught by Scott and Val, was about the tradeoff between the expressibility and the computational hardness of decisions in logic. Finally, Scott has generously allowed me to attend his Set theory course. I admit that I could not examine every corner of the vast field of study in logic covered in these courses, but if I know something about logic, then it is what I learned from these courses. I am deeply grateful to my professors.

The master's program at UPenn was not only about logic but a collection of lectures in different fields of computer science. I may not describe all the lessons and courses from the program, but I believe these courses have prepared me for any research I do now or in the future. Thus, I would like to thank some of the CIS department professors at UPenn-Jean Gallier, Sampath Kannan, Lyle Ungar, Michael Kearns, and Benjamin Pierce. Besides, I send my thanks to Pritham Choudhury, Yao Li, Yishuai Li, Xujie Si, Prathamesh Patil, and Rohan Ghuge for many illuminating discussions.

I am also indebted to my professors and friends from South Korea. Firstly, I am sincerely thankful to Jinyoung Choi, a professor at Korea University, whom I have considered my mentor since I was in college. I appreciate him for giving the most objective advice that I can hear on the subjects revolving around academia. I am particularly indebted to him for his advice and support in studying abroad, as he wrote a letter of recommendation for me. Furthermore, I am immensely grateful to him for introducing me to the theory of computation and logic that

has been planted in my mind as a seed that I kept and grew carefully with awe.

I have special thanks to Junkil Park, who guided me from even before I went to college until I went to UPenn. I knew him in the first place as a teacher who taught me programming and algorithms when I was young. After a while, I met him again when I went to Korea University as he was doing his Ph.D. with Jinyoung Choi. When I went to UPenn, I met him again since he was doing his Ph.D. there. I did not meet him in France, as he found a job in California, but sometimes, it feels like he will be somewhere out there, where I will be going! I know it was very fortunate to have such a perfect advisor like him, who gave much support and all practical advice in proximity.

I thank Sung Woo Jung, a Korea University professor, who gave me precious advice throughout my college studies. I am particularly grateful for his practical help and advice when I got interested in quantum computation, which was not a very available subject in South Korea around 2015. I am also indebted to Dong-joo Kim, another professor at the university, for the supervision of my internship in the brain and cognitive science. I thank him for bearing with me and putting much faith in me while I ambitiously put much effort and resource without producing any good results. I was grateful for his understanding when I quit the lab to pursue the call from logic and quantum computation. I am also infinitely grateful to Young-Tak Kim, whom I met in the same lab, who gave me any physical or moral support without hesitation.

Next, some friends of mine gave me huge influences. I am grateful to my friend, Seokje Cho, for letting me think about courage and teaching me by example that we can change our society. Despite its coercive appearance, the world has become fragile, and something requires much care after meeting him. I remember Juho Jung as an enthusiastic physicist. I have not seen anyone with as much excitement in physics as he does. I thank him for showing me the beauty of the physics he sees through exciting conversations. I also thank Yoobin Jeong for introducing me to the world of quantum physics through the book *The Emperor's New Mind* written by Roger Penrose. This introduction changed my question on how the brain works into the question on how the physical system works.

Last but not least, I would like to sincerely thank my parents, who have always supported me in whatever I do. I thank them for giving me their unconditional love. My parents cared for me far before my earliest memory and knew what I wanted to do even better than me. They have been happier than me whenever I had good news and sadder if I was disappointed. I know that I live the way they have taught and shown me, and their support was crucial for me to walk through it. Therefore, I would like to say that if I could pursue my goal despite difficulties and make some meaningful milestones, they also belong to theirs.

To conclude, I finally wrote thanks to everyone I wanted to thank. However, I know there will be people I will realize to whom I owe profound thanks. It is not because their help is less than the others' but because my memory is not perfect. Therefore, I send my sincere thanks to those I have not been able to recall at this moment. Without the help of all these people, I could not finish my thesis, and it is simply wondrous that I have always been encouraged by such friendly and capable people.

On thanksgiving day 2022,

Halifax, Canada

Titre: Méthodes Formelles pour les Langages de Programmation Quantiques

Mots clés: Programmation quantique, Méthodes formelles, Langage de programmation, Théorie des catégories

Résumé: Le modèle Qram est un modèle de calcul quantique pratique composé d'un ordinateur classique et un processus quantique qui communiquent entre eux. Le programme est exécuté sur l'ordinateur classique. Il envoie les instructions correspondant aux opérateurs quantiques sur le co-processeur, et reçoit le résultat de l'observation de l'état quantique. Ce modèle est considéré comme un modèle standard et plusieurs langages de programmation ont été conçus en basant sur ce modèle.

Alors que les programmes dans ce modèle sont capables de réaliser tout calcul quantique grâce à l'usage de la mémoire quantique, il est difficile de les analyser sans l'aide d'un autre ordinateur quantique. Ce problème suscite le besoin pour des méthodes formelles pour les langages programmations quantiques : les outils formels pour raisonner sur l'optimisation du code, pour l'analyser des ressources, et pour spécifier et prouver les propriétés des programmes quantiques.

La sémantique catégorique fait partie de ces méthodes qui fait le lien entre les opérateurs quantiques et les programmes et introduit le système logique qui peut décrire les propriétés sur l'état quantique dans le système de types. Bien que plusieurs travaux proposent des sémantiques catégoriques pour les langages

de description de circuits quantiques, aucun ne supporte l'usage du résultat d'une mesure au sein du processeur classique (le "levage dynamique").

Dans cette thèse, nous formalisons le levage dynamique qui transfère le résultat d'observations sur l'état quantique à l'information classique dans un langage de programmation de description de circuit quantique. En suivant l'approche du langage Proto-Quipper, nous définissons un langage typé de description de circuit quantique où l'information quantique levée est incorporée dans la structure ramifiée. Ensuite, le levage dynamique est formalisé dans le cadre de la sémantique opérationnelle et la sémantique catégorique.

Notre sémantique catégorique est basée sur le modèle de Francisco Rios et Peter Selinger pour le langage programmation Proto-Quipper-M. Pourtant, pour formaliser le levage dynamique, nous construisons une catégorie de Kleisli en capturant la mesure quantique comme un effet de bord sur une catégorie concrète pour le circuit avec la mesure quantique. Nous prouvons le théorème de correction de la sémantique catégorique par rapport à la sémantique opérationnelle.

Title: Formal Methods for Quantum Programming Languages

Keywords: Quantum programming, Formal Methods, Programming languages, Category theory

Abstract: The quantum random-access machine (QRAM) model is a practical model of quantum computation composed of a classical computer and a quantum processor communicating with each other. The program is executed on the classical computer. It can send instructions corresponding to quantum operations and receive measurement outcomes from the quantum co-processor. This model is expected to be the model of quantum computation in the near future, and a group of quantum programming languages has been developed based on it.

While the program in the model has the ability to simulate any quantum circuit with the help of a quantum processor, analyzing the program becomes difficult without relying on another quantum computer. This problem calls for the development of formal methods for quantum programming languages: formal tools to develop and reason on code optimization, analyze resources, and specify and prove the properties of quantum programs.

One of these tools is categorical semantics, which links the actions of quantum operators to the meaning of quantum programs and embeds logical systems on quantum states to the

type system of the language. Although categorical semantics for quantum programming languages is an established field, dynamic lifting—the ability to use the result of a measurement in the classical host—has so far only been considered in the context of denotational semantics based on operator algebras: a circuit in the model is not a syntactic object that can be manipulated.

In this thesis, we formalize dynamic lifting in a quantum circuit-description language which allows programs to transfer the result of measuring quantum data into classical data. Following the Proto-Quipper approach, we define a typed circuit-description language called Proto-Quipper-L, where the lifted data is encoded in the branching structure. Dynamic lifting is then formalized within the operational and categorical semantics.

Our categorical semantics is based on the model from Rios&Selinger for Proto-Quipper-M. However, to formalize dynamic lifting, we construct, on top of a concrete category of circuits with measurements, a Kleisli category, capturing the measurement as a side effect. We show the soundness of this semantics with regard to the operational semantics.

Contents

1	Introduction	17
2	Background	31
2.1	Quantum Computation	31
2.1.1	Quantum state	31
2.1.2	Quantum operators	31
2.1.3	Properties of quantum computation	33
2.1.4	Models of quantum computation	34
2.2	Example: Quantum teleportation	34
2.3	Lambda-calculus	35
2.4	Curry-Howard	37
2.5	Categorical semantics of Lambda-calculi	39
2.5.1	Categorical notions and constructions	40
2.5.2	Higher-order	44
2.5.3	Side-effects	47
2.5.4	Properties of categorical semantics	49
2.6	Linear logic	52
2.7	Semantics of linear logic	54
2.8	Lambda-Calculi with Linear Type	60
2.9	Approaches to Quantum Lambda-Calculi	61
2.9.1	Quantum Lambda-Calculi	61
2.9.2	Quantum Circuit Description Language	65
2.10	Categorical models of quantum computation	68
2.10.1	Completely positive maps	68
2.10.2	Semantics of quantum channels	70
2.10.3	Models of circuit-description languages	74
3	Type system and operational semantics of Proto-Quipper-L	83
3.1	Syntax of the language	84
3.1.1	Algebraic structure of quantum channel	85
3.1.2	Definition of the language	96
3.2	Type system	101
3.3	Operational Semantics	116
3.3.1	Circuit-buffering operational semantics	117
3.3.2	QRAM-based operational semantics	124
3.4	Type Safety Theorem	126
3.4.1	Type safety for circuit-buffering	126
3.4.2	Type safety for QRAM	138

4	Categorical Model	141
4.1	Category of quantum channels (i.e., diagrams)	143
4.1.1	Diagram	143
4.1.2	Category of diagrams and its product	145
4.1.3	Compact closed category of diagrams	155
4.2	Rios and Selinger completion	156
4.2.1	Definition of coproduct completion	157
4.2.2	Symmetric monoidal category $\overline{\overline{M}}$ from coproduct completion	158
4.2.3	Finite coproduct in $\overline{\overline{M}}$	161
4.2.4	Distributivity of \otimes over $+$	164
4.3	Benton's linear/non-linear category	166
4.3.1	!-Comonad	166
4.3.2	Symmetric monoidal comonad $(!, \epsilon : ! \rightarrow \mathbf{1}_{\overline{\overline{M}}}, \delta : ! \rightarrow !!)$ with $\pi_{A,B}$ and π_I	170
4.4	Lifting Monad	171
4.4.1	Definition of Branching computation monad (F, μ, η, t)	172
4.4.2	Kleisli category $\overline{\overline{M}}_F$ of the monad F	178
4.4.3	Natural isomorphism between $!F$ and $F!$	178
4.5	Interpretation of the language	179
4.5.1	Interpretation of quantum channel types and the circuit operators, Box and Unbox	181
4.5.2	Interpretation of the quantum channel constants	185
4.5.3	Interpretation of typing rules	189
4.5.4	Interpretation of the extended typing relation	209
4.5.5	Examples—non-trivial quantum channel and teleportation	210
5	Soundness	225
5.1	Preliminary lemmas	225
5.2	Basic type	226
5.3	Soundness theorem of the categorical semantics	228
6	Conclusion and Discussion	267
	Bibliography	271
A	Proof of the Preservation Theorem	281
A.1	Beta-reduction	282
A.2	Let-product elimination	287
A.3	If-then-else	294
A.4	Box	299
A.5	Unbox	305
A.6	Congruence: quantum channel	328
A.7	Congruence: on the right of application	331
A.8	Congruence: on the left of application	356
A.9	Congruence: on the left of a pair	373

A.10	Congruence: on the right of a pair	391
A.11	Congruence: if-then-else	407
A.12	Congruence: let-construct	422
A.13	Congruence: branching, case 1	442
A.14	Congruence: branching, case 2	447
A.15	Congruence: branching, case 3	452
A.16	Congruence: circuit reduction	457
B	Synthèse de la thèse	461
B.1	Contexte	461
B.2	Le Problème	461
B.3	Solution Proposée	462
B.4	Discussion et Ouverture	462

1 - Introduction

Quantum computation is interesting. The development of quantum mechanics (which was done by physicists like Max Planck, Niels Bohr, Albert Einstein, Werner Heisenberg, Erwin Schrödinger, et al.) [29] and the formalization of computation and logic (done by logicians and mathematicians like Alan Turing, Kurt Gödel, Bertrand Russell, Alfred North Whitehead, Alonzo Church, Gottlob Frege, David Hilbert, et al.) have created distinct research fields that have been very successful beginning from the period from late nineteenth century to the first half of the twentieth century. Although the results of each of the two fields had influences on the other field—e.g., the discovery of the transistor due to the understanding of solid-state electronics based on quantum mechanics [15] and the computational physics, which utilizes the high-performance computer and optimization methods to solve complex physical systems [37]—it was not until the late 1970s or 1980s that the central ideas of quantum mechanics, which are superposition of states and time evolution of states, have begun to be considered as an ingredient in the study of the model of computation [31, 19]. Conversely, it is also possible to consider the profound implication of logic in the theory of physics—like using topos theory in the formalization of quantum physics in [30]—but let us not delve into this subject since it is out of the scope of the thesis.

The introduction of these new quantum features as a resource of computation gave birth to interesting algorithms ranging from Deutsch-Jozsa algorithm [20] for the discrimination of different modes of input to Shor’s algorithm [66] for the factorization of large natural numbers. Moreover, the BB84 protocol [8], a secure quantum key distribution protocol, is another example of the advantages of using quantum resources. In these algorithms and protocols, new data types are introduced, which are called qubit, and the operations in process or functions are defined over qubits.

While quantum algorithms theoretically bring extraordinary advantages compared to conventional algorithms for specific problems, there have been doubts about whether these algorithms are realizable, i.e., whether we can make a practical quantum computer [49]. Even though there are multiple blueprints on the construction of quantum-based computers on different materials ranging from photons to trapped ions, each scheme has difficulties that makes it hard to build a universal quantum computer with many qubits. An example of such difficulties is the balance between the decoherence—which means the collapse of the quantum state induced by the interaction between the system and the environment—and the applicability of quantum operator—which requires an interaction between the system and the environment. This seemingly paradoxical requirement, i.e., to reduce decoherence while maintain-

ing the applicability of quantum operators, has been a good reason for many people to be skeptical about the development of real and practical quantum computers.

However, the situation has been changing recently with the development of quantum chips with more than fifty qubits by Google [5] and IBM [54]. In 2019, in its publication of the Sycamore processor, Google claimed the advent of quantum supremacy, which means the existence of a quantum process that the high-performance classical computer cannot simulate in a feasible time. Google and other companies plan to increase the number of qubits with the help of their scalable methods of construction of qubits.

Yet, it is worth noting that each qubit and quantum operation may not be perfect, meaning there can be errors. The fidelity of qubits and quantum operations may differ over the different realizations of qubits. Nevertheless, different error-correcting methods have been developed, starting with Shor's error-correction code [67], which would help us build reliable quantum computers even based on imperfect qubits and quantum operations.

With these recent successes, quantum computation or quantum informatics is gaining more interest worldwide while improving rapidly. Although we cannot say decisively yet that, for example, we will implement Shor's algorithm in a real quantum computer, it seems that we can afford the assumption that we will have a real and practical quantum computer someday. Given this, what we can do with a quantum computer becomes our next question.

What we can do with quantum computer A straightforward approach to answer the question is by showing problems that can be solved more efficiently by quantum computers than by classical computers. There are several examples of quantum algorithms: the list includes the quantum walks, the finding hidden-subgroup problem, quantum machine learning, graph isomorphism, ...These algorithms are classified by methods from which they obtain the quantum advantage over classical computation. Such methods include Quantum Fourier Transform (QFT), Phase Estimation (PE), and Quantum Walk (QW). However, in general, it is challenging to find a quantum algorithm that brings an advantage compared to classical computer since there are many problems that seem to be hard, but we do not know yet if there is a classical algorithm that efficiently solves the problem.

A more systematic approach to answering this question is formalizing quantum computation and classical computation and finding the difference between these models and its implications for algorithmic problems. A good example of the difference between the models of classical and quantum computation is contextuality. Intuitively, the contextuality of the quantum system is generated by the existence of non-commutative quantum operators and implies a contradiction together with the assumption of the existence of a global

state which maps a probability distribution to each operator. Let us elaborate on this contradiction since it is helpful to keep in mind the contextuality when discussing models of quantum and classical computation, and it is one of the primary sources of the weirdness of quantum logic that we would like to distinguish from classical logic in the future.

Let us assume that we have two parts of the system called A and B . Each part is equipped with two operators— p_A^1 and p_A^2 for A and p_B^1 and p_B^2 for B . Furthermore, we assume that each operator creates two possible outcomes 0 and 1. Now, we consider a state as a probability distribution over the possible outcomes of each operator. Since A and B are separated, we can apply one operator for each part simultaneously. For example, when the operators p_A^1 and p_B^1 are applied to A and B simultaneously, we obtain one among $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$ as outcome where the first and the second element of the pair represent the outcome of p_A^1 and p_B^1 , respectively. And a state is a probability distribution over the four outcomes.

Each of the following two tables shows possible states defined for each pair of operators applied simultaneously to the parts A and B .

$$\begin{pmatrix} & 0,0 & 0,1 & 1,0 & 1,1 \\ p_A^1, p_B^1 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ p_A^1, p_B^2 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ p_A^2, p_B^1 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ p_A^2, p_B^2 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{pmatrix} \quad \begin{pmatrix} & 0,0 & 0,1 & 1,0 & 1,1 \\ p_A^1, p_B^1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\ p_A^1, p_B^2 & \frac{2}{3} & 0 & \frac{1}{6} & \frac{1}{6} \\ p_A^2, p_B^1 & \frac{2}{3} & \frac{1}{6} & 0 & \frac{1}{6} \\ p_A^2, p_B^2 & \frac{1}{4} & \frac{1}{12} & \frac{1}{12} & \frac{1}{12} \end{pmatrix}$$

Both tables are obtained from actual quantum states. The left one is obtained from the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ and the right is from the state $\frac{1}{\sqrt{3}}(|00\rangle + |01\rangle + |10\rangle)$, where the operators p_A^1 and p_B^1 are the measurement over $|0\rangle$ and $|1\rangle$ basis while the operators p_A^2 and p_B^2 are the measurement over $|+\rangle$ and $|-\rangle$ basis.

The global state is then represented by a probability distribution over all 16 possible outcomes for each operator. This global state is supposed to generate the given table for each state by marginalization. However, it can be shown that there is no such global state for the table on the right while there exists a global state for the table on the left.

This example gives us a hint why we need a different model for quantum computation. Firstly, from the logical point of view (which is classically equivalent to the computational perspective), it is not straightforward to define quantum logic. For example, when we consider a quantum operator as a logical statement, the conventional model theoretical interpretation based on the valuation of logical sentences does not work since there may not be a global state. Secondly, from the computational perspective, contextuality shows that classical states are strictly included in quantum states. It can be considered resources that one can utilize to design algorithms that are impossible in classical computation. In addition, there is the limitation of quantum

computation, as well, in that not every table of the partial state is realizable as a quantum state.

Once we have formal models of classical and quantum computation, we can study their relationship. In one direction, we may try to find an embedding of classical functions into quantum functions, which would allow us to see the advantages of quantum computation as the extra parts in the model of quantum computation which has no counterpart in the model of classical computation. In the other direction, it is equally possible to think of a variant of the model of classical computation where the quantum function is introduced as the constant or black box. Given the equivalence between the model of classical computation and logic (by Curry-Howard isomorphism), this extension corresponds to the logic where the quantum function is specified in classical logic. This direction provides us with a way to verify quantum functions in classical logic.

This thesis partially concerns this question of the relationship between the models of classical and quantum computation: its initial goal was to build a toolchain of verifications of quantum programs, which can be viewed in line with the implication mentioned above on the verification of quantum programs. In particular, our approach to this goal relies on studying the semantics of programming languages and type systems of quantum computation. Remembering that λ -calculus was shown to formalize classical computation, namely, by Church-Turing thesis, we can hope that quantum programming languages (or quantum type systems) will formalize quantum computation where the execution of the quantum computation is represented as reduction, or equivalence, of terms. Combining these programming languages (or type systems) with classical logic where the quantum function is introduced as constant, we could obtain a toolchain for implementing and verifying quantum algorithms.

In this context, we would like to focus on the formalization of programming languages for quantum computation as the first step toward the formal verification of quantum algorithms. In particular, for practical reasons, we consider programming languages for a certain scenario where a classical computer controls a quantum processor. This scenario corresponds to the QRAM model of quantum computation, and it is considered to be the model of quantum computation for the near future, the so-called Noisy Intermediate-Scale Quantum (NISQ) era.

Current issues: how to write (to program) quantum algorithms, what is the meaning (the semantics) of quantum programs. So far, we have introduced the subjects of this thesis in a broad context, relating them to the general questions coming from computer science and logic. Now, let us delve into the

more precise subjects of the thesis—which are quantum programming languages and their categorical semantics.

Intuitively, a programming language explains how to represent computation, and its semantics represents the computation as a mathematical object. It has been studied for many years where one may consider Church’s λ -calculus as the first programming language whose semantics is given as complete lattices and Scott domains. It has grown into a big field as there are numerous programming languages for different purposes ranging from circuit description languages to probabilistic and statistic programming languages and programming languages for concurrent systems. Furthermore, it takes an important role in applications like software verification and testing, program synthesis, and type inference. In order to do that, the programming language is equipped with type systems and various semantics from which desired properties of a program can be represented and derived. Relying on a relationship between logic and type system, a program can be considered a proof and finding a proof of certain property becomes synthesizing a program.

These methodologies regarding the questions revolving around programming languages and formal semantics are called formal methods. Therefore, formal methods are the tools that we use to obtain practical results from the abstract definition of computation. In particular, in this thesis, we would like to use formal methods to decode and analyze quantum computation.

Programming language for quantum computation In quantum computation, one considers a special kind of memory where data is encoded into the state of objects governed by the laws of quantum mechanics. The basic unit for quantum data is the quantum bit, or qubit, and in general, a quantum memory is understood as consisting of individually addressable qubits. The state of a quantum memory can be represented by a unit vector in a complex Hilbert space. Elementary operations on qubits consist of unitary operations on the state space, called quantum gates, and measurements, which are probabilistic operations returning a classical boolean.

The usual model for quantum computation is the notion of quantum circuits. Quantum circuits are made of quantum gates and wires. A wire represents a qubit, and each gate, attached to one or several wires, is a unitary operation acting on the corresponding qubits. In this model, a computation consists of allocating a quantum register, applying a circuit (i.e., the list of gates, in order), followed by the measurement of the register to get a classical piece of information.

On the contrary, in the QRAM model, quantum computation is performed under the control of a classical host [34]. The classical host will emit a stream of

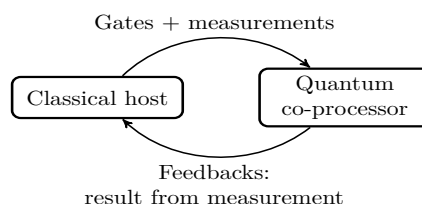


Figure 1.1: QRAM model

interleaved (pieces of) quantum circuits and measurements to the quantum co-processor. The quantum co-processor executes the instructions while returning the results of measurements on the fly to the classical host. This interaction between the host and the co-processor is illustrated in Figure 1.1. In this model, the computation is not a fixed linear list of quantum gates: the quantum gates emitted to the quantum co-processor depend on the result of intermediate measurements.

Various quantum programming languages, from quantum λ -calculus to Quipper and QWire, are based on this QRAM model. While quantum λ -calculus strictly follows the diagram of the QRAM model, Quipper and QWire provide circuit-level operations (like the reverse and control operators for the construction of quantum circuits and print and gate-count operators for the analysis of quantum circuits) which are applied to buffered quantum operations (consisting of gates and measurements). The features of classical programming language like high-order abstractions, branching statements, recursion, inductive data types, and dependent types have been studied and formalized in this context. These features provide a powerful and expressive way to represent quantum computation. However, it would be worth mentioning that there are features of quantum computation that do not have a classical counterpart, like quantum switch and indefinite causal order, which imply that functions, or sequences of functions, can form a superposed function, or an ensemble of functions. Finding and formalizing new features of quantum computation in programming languages constitutes an exciting part of the research of quantum programming languages.

Formal methods for quantum programming languages One of the motivations of formal methods is the verification of programs. As programs and systems that we deal with in real-world applications are very complicated, it is not straightforward anymore to decide whether the program is safe. By applying formal methods, a program, or some properties of the program, can be verified by the programming language's formal semantics and type systems. The same scenario applies to quantum programs. In particular, we need formal methods for quantum programs since it is not easy to get intuition on what the program does from a quantum program (imagine a sequence of unitary gates and measurements).

Formal methods for verification of programs are, in general, divided into two regimes which are static and dynamic analyses. Firstly, static analysis of a program means that the analysis is done without the execution of a program. It consists of, for example, a type system, abstract interpretation, and all analyses based on the semantics of programs. Next, dynamic analysis

constitutes the analysis of the program, which requires execution like testing. In practice, it is for now considered that running a program on a quantum processor requires more resources compared to a classical computer and that debugging a quantum programming is tricky because there is no way to inspect the quantum state directly, while the measurement causes the collapse of the quantum state. Therefore, formal methods applied to quantum programming language tend to be static analysis.

In particular, type theory has been commonly used in studying quantum programming languages to eliminate erroneous programs. In the context of quantum programming languages, the quantum data like qubit is considered not duplicable nor erasable, reflecting an implication of quantum mechanics. This property of quantum data is summarised by linearity which originated from linear logic.

In fact, type theory has a close relationship with logic, where the type represents the logical sentence, and the type inference rules correspond to the inference rules in the logic. Note that, in this context, a type system is a theory in certain logic whose universe of the model is the state of the program. As concrete examples, dependent type theory is related to intuitionistic first-order logic and simply-typed λ -calculus maps to intuitionistic propositional logic. Consequently, the analysis of the expressivity of a theory in logic can be applied to investigate the definability of properties in a type system. Therefore, one can conceive diverse type theories with different assumptions and different languages, which form different theories within the corresponding logic. To find out how to represent quantum properties and find a theory for reasoning about them would be an interesting question in the research of formal methods for quantum programming languages.

Indeed, there is a tradeoff between the expressivity of logic and the computational hardness of proving sentences in the logic. On the one hand, there can be a very expressive type theory where we can describe various properties in quantum computation. Considering that the type represents the program's state, this property may include any set in the powerset of the quantum states. As one can imagine, it is not trivial to prove a sentence in the logic, which is to find a program that transforms any state in the input type into some state in the output type. On the other hand, there can be a type theory where the expressivity is limited. For example, in a type system which has only qubits and their composition as quantum data type, type system is not expressive enough to distinguish two different quantum processes which take the same number of input and output qubits. However, this simplicity may allow us to automate the type checking process (finding typing derivations).

Therefore, it is left as a choice to select a proper type theory (the logic) and define the type constants and type inference rules (the language and the axioms in theory), depending on the purpose. In addition, this also falls into

the realm of abstract interpretation, where one finds abstract properties of states preserved through the computation.

Another example of formal methods is the formal semantics of programming language, which concerns the problem of finding the mathematical structure (or model) which characterizes programs represented by the programming language. If a type system describes the properties of program states, the semantics of programming languages attempts to characterize the meaning of each program (by finding a mathematical model which precisely represents the meaning of the program). There are two approaches to the problem, which are called operational and denotational semantics.

Firstly, operational semantics interprets each program as a sequence of the evaluation of the program. It can be considered as an abstract machine (like a Turing machine), where the program corresponds to machine instructions, which act on the program state (or a tape). Therefore, the configuration of the operational semantics is defined as a pair of the partially evaluated program and the program state. The execution of a part of the program is called reduction. Operational semantics gives intuitive formalization of classical and quantum computation.

However, the sequence of configurations from the operational semantics is hard to analyze. In particular, given that the sequence can be infinitely long, the equality of two programs is not decidable (by the reduction from the Halting problem). It is similar to the fact that although we describe the laws of physics by Newton's theory (which predicts the position and momentum of each particle in the system after an infinitesimal period), it is, in general, difficult to determine the exact position and momentum of particles after a long period.

Denotational semantics is another way to interpret the program, representing the program as a mathematical object preserved over the reduction derived by operational semantics. As the denotational semantics does not change throughout the execution, one can compare two programs by looking at the interpretation of any programs in the execution, including the initial programs. It is similar to the representation of particles as a path over time in Lagrangian mechanics, where each point of the path creates the same path. Denotational semantics is often related to categorical semantics, where the interpretation of a program is defined in a categorical notion called morphism.

In addition, note that for denotational semantics to be meaningful, one needs to make sure that the interpretation is equal for all programs in the reduction sequence defined by a given operational semantics. In general, a good denotational semantics requires that the interpretation is the same for all observationally equivalent programs (i.e., they can be replaceable in any context, in the sense that the whole program reduces to the same value). Moreover, another valuable property of denotational semantics is that the interpretation

distinguishes programs that are not observationally equivalent. Given these properties, denotational semantics gives a denotation of the program (or an equation theory of programs), which characterizes the meaning of the program given by the operational semantics. Continuing the analogy between the semantics of programming languages and the mechanics, these properties of denotational semantics could be considered analogous to the principle of stationary action in Lagrangian mechanics in that the principle provides criteria for selecting a particular path that minimizes the action among all paths given the initial and final states.

In summary, when the formal semantics is defined, the program is mapped to a mathematical object, and programs can be analyzed based on this interpretation.

Why are we interested in finding categorical semantics of programming languages? Category theory is often used as a denotational semantics of programming languages. However, it is hard to see how this abstract formalization helps us solve any practical problem, for example, verification of a program over some given specification and representation of specification. Therefore, we can ask why we are interested in categorical semantics and what it brings to us. In this paragraph, we would like to discuss what we can wish to do with the categorical semantics of programming languages.

The most straightforward reason to use category theory would be that it is a universal language that formally describes various objects in different fields, from mathematics and computer science to physics. Categorical theory can be firstly considered an abstract classification of different types of functions. Many things are functions; for example, time-dependent physical states like position and momentum are real or complex functions over time. Moreover, the valuation of the logical sentence is a Boolean function while, more generally, any logical formula is a function from the composition of types of all variables appearing in the formula to the type of the formula.

As a concrete example, quantum mechanics is formalized as the category of C^* -algebras—an object is a C^* -algebra, which refers to the Hilbert space of bounded operators; and a morphism is a $*$ -homomorphism between C^* -algebras, which refers to the quantum process. Then, the quantum state can be represented as a function over these bounded operators whose values represent probabilities. In this representation, a state should correspond to a $*$ -homomorphism from the C^* -algebra of operators to the C^* -algebra of probabilities whose meaning is applying the operator. Furthermore, as alluded to in the example of contextuality, a state of a system can also be considered a collection of probability distributions over possible outcomes assigned to each set of compatible operators. In this way, one can represent the collections of compatible operators by defining a category of the compatible operators of

C^* -algebras.

Another benefit of category theory is that it describes abstract objects. In fact, in the previous example of the category of C^* -algebras, the morphism is defined by the language of set theory, and it is not entirely category theory. The category defined by using the set-theoretic notion is called the concrete category. As we will discuss in detail, the simply-typed λ -calculus corresponds to a cartesian closed category which is an abstract definition of the category. Therefore, any concrete category that satisfies the requirements of the cartesian closed category can model the simply-typed λ -calculus. As a result, the abstract category lets us link seemingly different objects and, in some cases, use intuition on one object while studying the other object.

An example is the notion of symmetric monoidal category, which is a category with a symmetric and associative bifunctor, called tensor product, that has a unit. It can be interpreted as both a diagrammatic language and a category of vector spaces. A diagram consists of wires—the objects of a symmetric monoidal category—and boxes—the morphisms of the category. Then, the composition of morphisms gives a new diagram which is obtained by merging two diagrams that are given by the morphisms, while the bifunctor induces the juxtaposition of two diagrams. Next, in the category of vector spaces—whose objects refer to all vector spaces over a specific field and morphisms refer to all linear maps—the tensor product forms the tensor product of vector spaces and linear maps. Given this relationship, we can actually make a diagram of linear maps over vector spaces depending on how it is constructed in the category.

To sum up, from what we have discussed above, what we can expect to do with category theory is that we can find an abstract model of a programming language using the expressive language of category theory. Then, we can find different concrete categories that satisfy the abstract definition of the model. One of these concrete categories should be the model of quantum mechanics, which implies that each program in the programming language corresponds to a physical process and that any physical process can be represented as a program in the language.

Problem - mixing measurement and unitaries Now, let us introduce the specific features of quantum programming language that we want to formalize. Although quantum circuits and QRAM models are equivalent in expressive power, practical quantum computation is more likely to be based on the QRAM model. For this reason, many programming languages and their semantics are based on QRAM model [63, 47, 28, 77, 81, 72, 69]. An interesting implication of this model is that the quantum circuit construction in the classical host can be dependent on the result of measurement: there is a transfer of information from the quantum co-processor to the classical host. This feature is implemented, for example, in Quipper [28, 4] and QWire [47, 52]. Following

Quipper’s convention, we call this transfer dynamic lifting.

The classical control over the circuit construction imposed by dynamic lifting has not been explicitly formalized in the semantics of the circuit construction languages using it. To illustrate this problem, let us look at the program in Eq. (1.1). The program measures the qubit v_c and obtains the updated state of the qubit together with the resulting boolean b . Based on b , it then either allocates a new qubit initialized by true, then frees the qubit v_c , or simply returns v_c ¹. Despite this simple structure, the program does not correspond to a circuit because of the classical control.

$$\text{exp} ::= \text{let } \langle b, v_c \rangle = \text{meas}(v_c) \text{ in } \quad \text{if } b \text{ then } \langle \text{init}(\text{tt}), \text{free}(v_c) \rangle \text{ else } \langle v_c, * \rangle \tag{1.1}$$

In QWire, the operational semantics performs the normalization for composition and unbox operations, but the classical control by dynamic lifting is hidden in the host term within the unbox. In Quipper, the operational semantics is encoded in Haskell’s monadic type system and capture a notion of the dynamic circuit that includes measurements. However, this semantics has never been fully formalized.

Besides the operational semantics, programming languages for quantum circuits have been formalized using denotational semantics based on density matrices [47] and categorical semantics based on symmetric monoidal categories [61, 55, 38, 55, 24], or on the category of C^* -algebras [70, 53]. However, these examples of formalizations do not solve the problem in that they either ignore the structure of the circuit or keep the term with dynamic lift abstract. In particular, in [55, 24], the authors construct an expressive categorical model for the family of circuits (or parameterized circuit) and linear dependent type theory, respectively. However, they do not provide the semantics of dynamic lifting explicitly.

Outline of the work - add dynamic lifting in formalized quantum programming languages Our goal in this thesis is to find a model and to formalize the semantics for interleaved quantum circuits and dynamic lifting. The problem rests on how to analyze the structure of the computation without requiring the quantum co-processor to settle the values of measurements. We solve the problem by making circuits not only lists but trees branching over the result of measurements: we call such objects quantum channels. Hence, the semantics of dynamic lifting can be formalized by the generation of multiple control flows in the classical computation, each of which is interpreted as a quantum channel.

¹The program actually returns a pair consisting of a qubit and the unit term $*$ so that it is well-typed. We assume that the return type of free is the unit.

In this thesis, we propose both a small-step operational semantics and a categorical semantics for a typed language extending quantum lambda calculus [63] with circuit construction operators (box and unbox) and circuit constants. The formalization extends the one of Proto-Quipper [56]: circuits are generalized to quantum channels enabling the formalization of dynamic lifting semantics. A quantum computation that only consists of unitary gates deterministically reduces to only one possible value. On the other hand, a quantum computation using dynamic lifting might reduce to different values depending on the results of the measurements. To support this, the language is extended with a notion of branching terms, representing the non-determinism of computations. We prove a type safety theorem that ensures that a well-typed term does not get stuck and that types are preserved over reduction.

Next, we propose a candidate for a sound categorical model for the language. The model is based on the co-product completion of a symmetric monoidal closed category which is introduced in [55]. This category is used initially to separate parameters (which is known at circuit generation stage) and the (high-order) states (which require the execution of the circuit), where the states can be parameterized by parameters but not vice versa. However, one can notice that the same model could be used to formalize branching terms and quantum channels: use the path of a term in a branching term as a parameter and the term itself as a state, and, similarly, the path in the quantum channel as a parameter and the quantum state as state. Moreover, the non-deterministic computation which creates a branching term can be modeled by a strong monad over the category as shown in [44, 76].

We realize this idea of categorical semantics by defining a concrete category of diagrams \overline{M} , which is a symmetric monoidal closed category with a product. Then, we obtain the co-product completion $\overline{\overline{M}}$ of the category. On top of it, we define a strong monad over $\overline{\overline{M}}$ which maps a circuit family parameterized by a set to another circuit family parameterized by the multi-set of the set. Based on this structure, we provide the interpretation of the type system into the Kleisli category of the monad and show weak soundness of the semantics.

Specifically, the weak soundness theorem states that the interpretation is preserved over some reduction of typing derivation, based on the operational semantics. However, as some typing judgments admit more than one typing derivation, it does not imply that the interpretation is preserved over the operational semantics. Consequently, we can only say that any typing derivations that reduce to the same typing derivation are denotationally equal. Nevertheless, together with the fact that the operational semantics reduces any term into a value and that there is a unique typing derivation for some basic types, we can conclude that, for those basic types, all type judgments have the same interpretation and the interpretation is sound.

Outline of the thesis In this thesis, we introduce dynamic lifting into the Proto-Quipper language from Neil Ross’s Ph.D. thesis [56]. Dynamic lifting allows a program to transfer quantum data, qubit, into classical data, boolean. However, this process introduces non-determinism without the simulation of quantum circuits. To formalize dynamic lifting:

- first, the language is extended with branching terms, which represent the non-determinism of states of computation;
- second, quantum circuits are extended to quantum channels, which are trees of quantum circuits where each measurement creates a branch;
- third, the operation of dynamic lifting is encoded as a quantum channel constant, called *meas*, which creates a branching term for each boolean value from a given a qubit.

We define the type systems and the operational semantics of the language and show safety properties of the type system.

Next, we introduce categorical semantics for the proposed language, which is based on the co-product completion of a monoidal closed category from Francisco Rios and Peter Selinger’s paper [55]:

- first, we define a concrete category of diagrams, \overline{M} , and show that it is a symmetric monoidal closed category with a product;
- next, we define the co-product completion of \overline{M} , which is called $\overline{\overline{M}}$ as in [55].

On top of $\overline{\overline{M}}$, we define a symmetric monoidal strong monad $(F : \overline{\overline{M}} \rightarrow \overline{\overline{M}}, \mu, \eta, t)$ which models non-deterministic computations. Then, type derivations are interpreted as morphisms of the Kleisli category given by the monad F . Finally, we show a weak version of the soundness of the interpretation and the uniqueness of the typing derivation for basic types.

2 - Background

2.1 . Quantum Computation

This section provides a quick recap of what is needed about quantum computation for this thesis. Although quantum computation is inspired by physics, we rely on the mathematical formalism presented and used by Peter Selinger in [59]. A complete introduction to quantum computation can be found in, for example, [45].

2.1.1 . Quantum state

The state of a qubit is defined as the set of normalized vectors in a (finite dimensional) vector space over complex number. When we equip the vector space with a basis, for example, the computational basis, the state can be represented as normalized vector $q = \alpha |0\rangle + \beta |1\rangle$, meaning that $|\alpha|^2 + |\beta|^2 = 1$, where $\{|0\rangle, |1\rangle\}$ is the orthonormal basis called computational basis.

The joint state-space of two systems is the tensor product of the two-state spaces. If $\{|i\rangle_A\}_i$ is a basis for the state-space of the system A and $\{|j\rangle_B\}_j$ is a basis for the state-space of system B , then the $\{|ij\rangle_{AB}\}_{i,j}$ is a basis for the state-space of joint system $A \otimes B$.

For instance, the state of two qubits is in general

$$\alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle \quad (2.1)$$

with $\sum |\alpha_{ij}|^2 = 1$.

Note that there are states that are not separable, i.e., entangled states, which means that a state cannot be represented as a tensor product of vectors from separated state spaces. For example, the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ cannot be written as $|\phi\rangle \otimes |\psi\rangle$ of two qubits $|\phi\rangle$ and $|\psi\rangle$.

2.1.2 . Quantum operators

Next, we introduce quantum operations over the quantum states. There are three kinds of operation—initialization, unitary maps, and measurements.

Initialization

First, an initialization is a unit vector in qubit and we can define a linear map from the one-dimensional vector space, or scalar, to the subspace created by the unit vector in qubit that preserves the norm. In particular, we give an instance of initialization for each element in the computational basis of qubit ($|0\rangle$ and $|1\rangle$), namely,

$$\text{init}_{|0\rangle} = |0\rangle \text{ and } \text{init}_{|1\rangle} = |1\rangle.$$

Unitary maps

A unitary map is a linear map over a state space of a particular dimension that preserves normalized vectors. A unitary map can be represented in the following form.

$$U ::= \sum_{i,j \in B} u_{i,j} |j\rangle \langle i|$$

where $B = \{|0\rangle, \dots, |n\rangle\}$ represent the basis of the state space. It describes the meaning of the unitary map in terms of change of basis, where each element in the basis is mapped to a linear combination of the basis. Then, the new basis needs to be orthonormal.

Equivalently, one can represent a unitary map as a matrix.

$$U = \begin{pmatrix} u_{0,0} & \cdots & u_{0,n} \\ \vdots & \ddots & \vdots \\ u_{n,0} & \cdots & u_{n,n} \end{pmatrix}$$

Then, the unitary condition can be represented in the linear algebraic equation $UU^* = I = U^*U$, where U^* is the complex conjugate of U .

Although any unitary maps are allowed in quantum computation, we often restrict ourselves to a universal set of unitary maps since any unitary map can be approximated, up to an arbitrarily small error, as a composition of unitary maps from the universal set. An example of a universal set is:

$$N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, V = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, W = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{i} \end{pmatrix},$$
$$N_c = \begin{pmatrix} I & 0 \\ 0 & N \end{pmatrix}, H_c = \frac{1}{\sqrt{2}} \begin{pmatrix} I & 0 \\ 0 & H \end{pmatrix}, V_c = \begin{pmatrix} I & 0 \\ 0 & V \end{pmatrix}, W_c = \begin{pmatrix} I & 0 \\ 0 & W \end{pmatrix},$$
$$X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where the first four maps are unitary maps over one qubit, and the rest are unitary maps over two qubits.

Measurement

For this thesis, we allow measurement over the computational basis on each qubit. Measurement is basically an observation of a quantum state that collapses the quantum state at the same time. Firstly, the observation of quantum state provides information on quantum state represented by classical data like Boolean value. Even if we measure the same quantum state, we may obtain different values, which forms a probability distribution characterized by

the quantum state. Secondly, the collapse of the quantum state can be considered a projection of the vector onto a subspace that depends on the observation value.

Formally, measurement on a qubit, $\text{meas} = ([|0\rangle], [|1\rangle])$, is defined as a family of projection spaces, $[|0\rangle]$ and $[|1\rangle]$, indexed by observation value, 0 and 1. When we apply measurement on a quantum state represented as $q = \alpha|0\rangle + \beta|1\rangle$, then we will observe 0 with the state in the projection space $|0\rangle$ with probability $|\alpha|^2$ and observe 1 with the state in the projection space $|1\rangle$ with probability $|\beta|^2$. Note that the state is normalized after the measurement.

Next, over the quantum state of multiple qubits indexed by a set I , measurement meas_i over the qubit $i \in I$ is defined as a family of projection spaces, joint state-spaces of qubits except for i where the state-space of qubit i is either the subspace $|0\rangle$ or $|1\rangle$, indexed by observation value, 0 and 1. When we apply measurement meas_i on a quantum state represented as $q = \sum_{\vec{a} \in \{0,1\}^I} \alpha_{\vec{a}} |a\rangle$, then we will observe 0 with the state in the projection space $[|a \in A_0\rangle]$, where A_0 is the subset of $\{0,1\}^I$ whose i -th element is 0, with probability $\sum_{a \in A_0} |\alpha_a|^2$; and observe 1 with the state in the projection space $[|a \in A_1\rangle]$, where A_1 is the subset of $\{0,1\}^I$ whose i -th element is 1, with probability $\sum_{a \in A_1} |\alpha_a|^2$. For example, when we measure the quantum state in Eq. (2.1) on the first qubit, we will observe the value 0 with the state $\frac{1}{|\alpha_{00}|^2 + |\alpha_{01}|^2} (\alpha_{00}|00\rangle + \alpha_{01}|01\rangle)$ with probability $|\alpha_{00}|^2 + |\alpha_{01}|^2$; and observe the value 1 with the state $\frac{1}{|\alpha_{10}|^2 + |\alpha_{11}|^2} (\alpha_{10}|10\rangle + \alpha_{11}|11\rangle)$ with probability $|\alpha_{10}|^2 + |\alpha_{11}|^2$. Measurements on different qubits are commutative; which means that we will observe the same value with the same state with the same probability for both cases—where we measure the qubit i and then measure the qubit j and where we measure the qubit i after that we measure the qubit j .

Although we can consider measurement as a projection operator whose projection space is chosen probabilistically, one can also consider measurement from a global point of view, i.e., measurement creates a probability distribution of quantum states. It leads us to generalize the quantum state, in which case our definition of the quantum state is called a pure state, and the probability distribution of quantum states is called a mixed state.

2.1.3 . Properties of quantum computation

Superposition and entanglement of states are properties of the quantum computation that distinguish it from the classical computation. Superposition describes that the quantum state is the linear combination of the computational basis corresponding to the classical state. Moreover, by applying unitary maps, one can transform one quantum state into another while classical computation (e.g., lambda-calculus) corresponds to permutation unitary maps. It seems that quantum computation gives a considerable advantage over classical computation.

However, there are some restrictions in quantum computation based on the transformation of the quantum state. First of all, quantum operators still need to be local; otherwise, it would mean that information can be transferred faster than light. In other words, although the quantum operator includes all unitary maps over the quantum state, each unitary map should be constructed by applying unitary maps over the quantum state of a finite number of qubits. It does limit the advantage of quantum computation, particularly when we only allow unitary maps over one qubit, in which case, all states are separable (we cannot create an entangled state), and separable state can be efficiently encoded in classical computation. In effect, entanglement in a quantum state is considered a real quantum resource that classical computation cannot have, and many quantum algorithms use entanglement to get a quantum advantage.

Secondly, although quantum states are much larger than the classical states and unitary maps serve as a powerful tool for computation, there is a problem with obtaining (classical) information from quantum states. There are several tasks that quantum mechanics prohibits. For example, no-cloning [80] theorem states that a quantum state cannot be duplicated. This theorem is a concrete example that distinguishes quantum computation from the classical computation. Many programming languages for quantum computation, including the one discussed in this thesis, need a particular typing system that excludes programs that violate this property.

2.1.4 . Models of quantum computation

Quantum circuit

A Quantum circuit is a model of quantum computation which consists of wires representing qubits and boxes for unitary maps. Therefore, the quantum circuit can be thought of as a big unitary map, and a quantum circuit is obtained by composing quantum circuits horizontally and vertically. Sometimes, it is assumed that each qubit is measured after the application of the circuit.

Quantum channel

Although pure quantum computation can be represented as a quantum circuit, the model of quantum computation on which we rely in this thesis is going to be the quantum co-processor model—aka QRAM model—[35]. In this model, the classical host accesses the quantum state through the quantum co-processor by sending quantum operators and receiving the quantum co-processor’s response for each measurement. We call a sequence of quantum operations mixing unitary maps and measurements a quantum channel.

2.2 . Example: Quantum teleportation

Quantum teleportation is an exemplary quantum protocol [45] where quantum information is transferred through classical data and entangled qubits. In

the protocol, which is illustrated in Figure 2.1, we assume that each of two remote parts, called A and B , share one qubit of a Bell pair: let us call the qubit that A takes y and the qubit that B takes q . The part A , given a qubit, x of any state, measures the qubits x and y then transfers the classical measurement results to B . The part B then applies different unitary operations on q depending on classical data from the part A . The resulting state of q and the unknown state of x can be shown to be the same.

Quantum teleportation has been a recurring example to several works on quantum protocols [1] and quantum programming languages [46] because it requires both quantum and classical data. For example, in our example, the part A receives two qubits and returns two classical bits, while the part B returns one qubit when given two classical bits and one qubit. Quantum teleportation will be a driving example in this paper, and we will show how the protocol can be expressed in Proto-Quipper-L and interpreted into a graphical language.



Figure 2.1: Quantum teleportation

2.3 . Lambda-calculus

Lambda-calculus [7] is a versatile model of higher-order programming languages, where functions are first-order terms that can be returned or passed along as arguments. The syntax of lambda calculus is defined in Eq. 2.2.

$$\text{(syntax of } \lambda\text{-calculus)} \quad t, t_1, t_2 ::= x \mid \lambda x.t \mid t_1(t_2) \quad (2.2)$$

where x refers to variable, $\lambda x.t$ is the abstraction of the λ -term t , and $t_1(t_2)$ refers to the application of the term t_1 to t_2 . Note that the formalism of lambda-calculus can easily be extended with pairing, injections, and matching, side-effects: probabilistic or non-deterministic behavior, inputs/outputs, global state, etc. An example of λ -term is Church numerals and arithmetic operations (e.g., $+$) over the Church numerals, which allows us to represent arithmetic expressions in lambda-calculus. First, we define Church numerals to represent natural number as in Eq. 2.3.

$$\bar{n} ::= \lambda f.\lambda x.f^n(x) \quad (2.3)$$

According to the definition, we can derive some instance examples of the terms of natural numbers as follows.

$$\begin{aligned}\bar{0} &= \lambda f. \lambda x. x \\ \bar{1} &= \lambda f. \lambda x. f(x) \\ \bar{2} &= \lambda f. \lambda x. f(f(x)) \\ &\vdots\end{aligned}$$

Then, we can define successor function S and addition $+$ as follows.

$$\begin{aligned}S &= \lambda n. \lambda f. \lambda x. f((n(f))(x)) \\ + &= \lambda n_1. \lambda n_2. \lambda f. \lambda x. (n_1(f))((n_2(f))(x))\end{aligned}\tag{2.4}$$

In order to see that these operations are indeed the operations that we expect, we need to be able to execute the lambda terms. In specific, what we want to show is that

$$\begin{aligned}S(\bar{n}) &= S(\lambda f. \lambda x. f^n(x)) \\ &\stackrel{?}{=} \lambda f. \lambda x. f^{n+1}(x) \\ &= \overline{n+1} \\ +(\bar{n}_1, \bar{n}_2) &= (+(\lambda f. \lambda x. f^{n_1}(x)))(\lambda f. \lambda x. f^{n_2}(x)) \\ &\stackrel{?}{=} \lambda f. \lambda x. f^{n_1+n_2}(x) \\ &= \overline{n_1+n_2}\end{aligned}\tag{2.5}$$

The equation with question marks needs to be shown by the equivalence relation over the terms.

Lambda-calculus comes with an equivalence theory which consists of α -equivalence and β -equivalence. The α -equivalence states that terms with different bounded variable are equivalent, i.e. $(\lambda x. t = \lambda y. t[y/x])$ where $t[y/x]$ means the term obtained by renaming the variable x by y in term t . The β -reduction states that the application of an abstracted term to a term is equivalent to the substitution of the variable by the second term in the first abstracted term, i.e. $((\lambda x. t_1)(t_2) = t_1[t_2/x])$. These rules are contextual, meaning that we can substitute a term with an equivalent term in a larger term.

With the help of the equation theory, we can derive the equation with question marks in Eq. 2.5 as follows:

$$\begin{aligned}S(\lambda f. \lambda x. f^n(x)) &= (\lambda n. \lambda f. \lambda x. f((n(f))(x)))(\lambda f. \lambda x. f^n(x)) \\ &= \lambda f. \lambda x. f(((\lambda f. \lambda x. f^n(x))(f))(x)) \\ &= \lambda f. \lambda x. f((\lambda x. f^n(x))(x)) \\ &= \lambda f. \lambda x. f(f^n(x)) \\ &= \lambda f. \lambda x. f^{n+1}(x)\end{aligned}$$

and

$$\begin{aligned}
& (+(\lambda f.\lambda x.f^{n_1}(x)))(\lambda f.\lambda x.f^{n_2}(x)) \\
& =((\lambda n_1.\lambda n_2.\lambda f.\lambda x.(n_1(f))((n_2(f))(x)))(\lambda f.\lambda x.f^{n_1}(x)))(\lambda f.\lambda x.f^{n_2}(x)) \\
& =(\lambda n_2.\lambda f.\lambda x.((\lambda f.\lambda x.f^{n_1}(x))(f))((n_2(f))(x)))(\lambda f.\lambda x.f^{n_2}(x)) \\
& =\lambda f.\lambda x.((\lambda f.\lambda x.f^{n_1}(x))(f))(((\lambda f.\lambda x.f^{n_2}(x))(f))(x)) \\
& =\lambda f.\lambda x.((\lambda f.\lambda x.f^{n_1}(x))(f))((\lambda x.f^{n_2}(x))(x)) \\
& =\lambda f.\lambda x.((\lambda f.\lambda x.f^{n_1}(x))(f))(f^{n_2}(x)) \\
& =\lambda f.\lambda x.(\lambda x.f^{n_1}(x))(f^{n_2}(x)) \\
& =\lambda f.\lambda x.(f^{n_1}(f^{n_2}(x))) \\
& =\lambda f.\lambda x.(f^{n_1+n_2}(x))
\end{aligned}$$

Note that there are choices that we have made in this example, i.e., there are multiple subterms to apply the β -reduction rule. These choices comes from the facts that the term with application $t_1 t_2$ creates two subterms t_1 and t_2 each of which may reduces to some other terms t'_1 and t'_2 and that if t_1 is an abstraction, then we can apply β -reduction rule at anytime. It leads us to two standard reduction strategies: call-by-name, which first applies the β -reduction rule before reducing the subterm t_2 ; and call-by-value [48], which reduces the subterm t_2 first.

Although the confluence property of lambda-calculus guarantees that each term reduces to the same term in the end regardless of choice if the evaluation terminates, different reduction strategies can produce different evaluation results in some extensions of lambda-calculus. For example, with side-effects, the behavior of a term might depend on the choice of reduction strategy.

Now, let us give an example of lambda calculus where different reduction strategies make different evaluations. In specific, consider the term

$$(\lambda x.\langle x, x \rangle)(a + b)$$

where $\langle M, N \rangle$ stands for pairing and $+$ stands for non-deterministic choice. In a call-by-value strategy this term reduces to $\langle a, a \rangle + \langle b, b \rangle$ while in call-by-name it yields $\langle a, a \rangle + \langle a, b \rangle + \langle b, a \rangle + \langle b, b \rangle$.

If some programming languages such as Haskell [33] follow a call-by-name strategy (technically, call-by-need), other languages [21, 3] choose call-by-value for its arguably more natural behavior for the programmer.

2.4 . Curry-Howard

Lambda-terms can be typed [6], and this forms the basis for the Curry-Howard isomorphism [17, 18, 32], driving a correspondence between programming languages and logical systems. In this context, a programming language is defined in two layers: a language of programs (or terms) and a type system,

which consists in the definition of types and type derivation rules. Similarly, a logic, in sequent calculus style, is defined by the definition of a sequent, which consists of sentences, and derivation rules. The isomorphism is then based on a map between types and sequents and a map between type derivations and proofs. Furthermore, once we interpret a type derivation as a proof of a sequent, then the proof normalization, which is based on the equivalence of proofs gives a notion of computation of the derivation.

As an example, let us consider the simply typed λ -calculus with conjunction \wedge and disjunction \vee , defined below in Definition 2.4.1.

Definition 2.4.1. Terms and types of simply typed lambda calculus with conjunction and disjunction are defined as follows.

$$\begin{aligned}
(\text{term}) \quad t, t_1, t_2 & ::= v \mid c^T \mid \lambda x : A.t \mid t_1 t_2 \mid \langle t_1, t_2 \rangle \mid p_1(t) \mid p_2(t) \mid \\
& \quad i_1(t) \mid i_2(t) \mid \mathbf{match}(x = t).(t_1, t_2) \\
(\text{type}) \quad A, A_1, A_2 & ::= \perp \mid T \mid A_1 \rightarrow A_2 \mid A_1 \wedge A_2 \mid A_1 \vee A_2
\end{aligned}$$

where T a base type.

In the term language, we let v be variable and c^T be constant of type T . Moreover, one can construct an abstract term $\lambda x : A.t$, which represents a term t depending on a variable x of type A , and an application $t_1 t_2$, which means application of t_1 to t_2 . Furthermore, a pair $\langle t_1, t_2 \rangle$ and the projections $p_1(t)$ and $p_2(t)$ can be constructed from any terms t_1, t_2 , and t . Similarly, the injections $i_1(t)$ and $i_2(t)$ and the match statement $\mathbf{match}(x = t).(t_1, t_2)$ (whose meaning is the substitution either $t_1[t'/x]$ or $t_2[t'/x]$ depending on t which is either $i_1(t')$ or $i_2(t')$) from any terms t_1, t_2 , and t .

For the type language, type can be a constant \perp or T , and one can construct function type $A_1 \rightarrow A_2$ and conjunction and disjunction types $A_1 \wedge A_2$ and $A_1 \vee A_2$.

Next, the type judgment has the form $\Gamma \vdash t : A$ where Γ consists of a set of pairs of a variable and a type, and t and A are a term and a type. Then, the type derivation rules for simply typed λ -calculus are given in Definition 2.4.2.

Definition 2.4.2. The type derivation rules for simply typed λ -calculus are as follows.

$$\begin{array}{c}
\frac{}{\Gamma, x : A \vdash x : A} \quad \frac{\Gamma \vdash t : \perp}{\Gamma \vdash t : A} \quad \frac{}{\Gamma \vdash c^T : T} \\
\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : B} \quad \frac{\Gamma \vdash t_1 : A \rightarrow B \quad \Gamma \vdash t_2 : A}{\Gamma \vdash t_1 t_2 : B} \\
\frac{\Gamma \vdash t_1 : A \quad \Gamma \vdash t_2 : B}{\Gamma \vdash \langle t_1, t_2 \rangle : A \wedge B} \quad \frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash p_1(t) : A} \quad \frac{\Gamma \vdash t : A \wedge B}{\Gamma \vdash p_2(t) : B}
\end{array}$$

$$\begin{array}{c}
\frac{\Gamma \vdash t : A}{\Gamma \vdash i_1(t) : A \vee B} \quad \frac{\Gamma \vdash t : B}{\Gamma \vdash i_2(t) : A \vee B} \\
\frac{\Gamma \vdash t : A_1 \vee A_2 \quad \Gamma, (x : A_1) \vdash t_1 : B \quad \Gamma, (x : A_2) \vdash t_2 : B}{\Gamma \vdash \mathbf{match}(x = t).(t_1, t_2) : B}
\end{array}$$

On the other hand, intuitionistic propositional logic is defined as in Definition 2.4.3.

Definition 2.4.3. Inference rules of intuitionistic propositional logic in sequent calculus form are presented below.

$$\begin{array}{c}
\frac{}{\Gamma, A \vdash A} \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash A} \quad \frac{}{\Gamma \vdash T} \\
\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \quad \frac{\Gamma \vdash A \rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \\
\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \\
\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \quad \frac{\Gamma \vdash A_1 \vee A_2 \quad \Gamma, A_1 \vdash B \quad \Gamma, A_2 \vdash B}{\Gamma \vdash B}
\end{array}$$

By considering a type as a proposition, we can see that the type derivation rules give the derivation of the sequent in the intuitionistic propositional logic and see that the term represents the proof, i.e., the typing derivation.

2.5 . Categorical semantics of Lambda-calculi

Category theory is a formalism well-suited to describe structures, and it has successfully served as a backbone for the semantics of programming languages [71, 2].

Among multiple definitions of a category, one can define a category as a directed (multi) graph whose nodes are called objects and edges are called morphisms and which is subject to several conditions. First of all, all objects and morphisms have names—we use capital letters A, B, \dots to denote the objects and small letters f, g, \dots to denote the morphisms. Then, a category has a morphism named $\text{id}_A : A \rightarrow A$ (or identity morphism) from A to itself, for each object A . Moreover, for any two morphisms $f : A \rightarrow B$ from the object A to B and $g : B \rightarrow C$ from B to C , there exists a morphism $g \circ f : A \rightarrow C$. Then, category satisfies the following axioms:

$$f \circ \text{id}_A = f = \text{id}_B \circ f \tag{2.6}$$

$$h \circ (g \circ f) = (h \circ g) \circ f \tag{2.7}$$

for any objects A, B, C , and D , and morphisms $f : A \rightarrow B$, $g : B \rightarrow C$, and $h : C \rightarrow D$. The first axiom states that the composition of a morphism with the morphism id , i.e., identity morphism, is the morphism itself and the second axiom states that the composition of morphisms is associative.

When interpreting a language in a category, a type A is mapped to an object $\llbracket A \rrbracket$ and a typing judgement $x_1 : A_1, \dots, x_n : A_n \vdash M : B$ to a morphism $\llbracket A_1 \rrbracket \times \dots \times \llbracket A_n \rrbracket \rightarrow \llbracket B \rrbracket$, assuming that the category has product \times structure. The interpretation of each typing rule informs us how to construct a morphism from the morphisms constructed from the typing derivation of subterms in the category from the semantics. Therefore, when we consider a category as a graph, the interpretation process transforms a typing derivation into a path in the graph of the category.

A question, then, arises when we try to interpret the language—how to encode different typing rules in terms of categories or categorical structure. In order to do that, we assume that the category exhibits particular structures—to name a few, the cartesian closed category or the monoidal closed structure. These structures provide us with objects and morphisms in the category, named in particular ways. These structures are defined using categorical constructions like cartesian category, functors, natural transformations, and adjointness, which will be summarized in this section.

These constructions are defined for any category, and their conditions are set as axioms for the model that interprets the language. However, sometimes, categorical semantics are defined over some other mathematical structure like Sets, in which case, the category is called concrete, and the axioms of the categorical construction should be derived for the underlying mathematical structure. In this case, the categorical semantics serves as an intermediate layer in the interpretation process from the language and the type system to the mathematical structure.

2.5.1 . Categorical notions and constructions

Now, let us introduce some categorical notions and constructions that appear in the interpretation of lambda-calculus.

Commuting diagram

To say that two morphisms are equal, we often use the notion of a commuting diagram. When we say that a diagram commutes, all morphisms of the paths from an object to another object in the diagram are equal.

Functor

A functor $F : \mathcal{C} \rightarrow \mathcal{D}$ from a category \mathcal{C} to a category \mathcal{D} is a map which sends each object A in \mathcal{C} to the object $F(A)$ in \mathcal{D} and which sends each morphism $f : A \rightarrow B$ in \mathcal{C} to morphism $F(f) : F(A) \rightarrow F(B)$ in \mathcal{D} which preserves the identity morphism ($F(\text{id}_A) = \text{id}_{F(A)}$) for any morphism A and

the composition of morphisms $(F(g \circ f) = F(g) \circ F(f))$ for any objects $A, B,$ and C and morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$.

Natural transformations

A natural transformation $\alpha : F \rightarrow G$ from a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ to a functor $G : \mathcal{C} \rightarrow \mathcal{D}$ maps each object A in \mathcal{C} to a morphism $\alpha(A) : F(A) \rightarrow G(A)$ in \mathcal{D} which satisfies the naturality condition, which is, for any morphism $f : A \rightarrow B$ in \mathcal{C} , the following diagram commutes:

$$\begin{array}{ccc} F(A) & \xrightarrow{\alpha(A)} & G(A) \\ \downarrow F(f) & & \downarrow G(f) \\ F(B) & \xrightarrow{\alpha(B)} & G(B) \end{array}$$

In other words, it means that $\alpha(B) \circ F(f) = G(f) \circ \alpha(A)$. The composition of natural transformations is defined by the composition of morphisms of category \mathcal{D} at each object in \mathcal{C} .

Functor category

Given two categories \mathcal{C} and \mathcal{D} , all functors from \mathcal{C} to \mathcal{D} and all natural transformations between the functors form a category called functor category, $\mathcal{D}^{\mathcal{C}}$. In specific, each functor $\mathcal{C} \rightarrow \mathcal{D}$ forms an object and each natural transformation from functor F to G forms a morphism. The identity is given by the identity natural transformation, which maps each object in \mathcal{C} to identity morphism in category \mathcal{D} . The composition of natural transformations is the composition of two morphisms.

Product of categories

Given two categories \mathcal{C} and \mathcal{D} , the product of these categories is the category $\mathcal{C} \times \mathcal{D}$ with an object (A, B) for each object A in \mathcal{C} and object B in \mathcal{D} and a morphism $(f, g) : (A, B) \rightarrow (A', B')$ for each morphism $f : A \rightarrow A'$ in \mathcal{C} and morphism $g : B \rightarrow B'$ in \mathcal{D} . This category can be shown to satisfy the axioms of the category with the identity morphism defined by the pair of identity morphisms of \mathcal{C} and \mathcal{D} and the composition of morphisms defined by the compositions of morphisms in the same categories.

Adjointness of functors

Definition 2.5.1 (Definition of adjointness (Lambek J. and Scott P.J.)). An adjointness between two categories \mathcal{C} and \mathcal{D} is defined as quadruple (F, G, η, ϵ) where $F : \mathcal{C} \rightarrow \mathcal{D}$ and $G : \mathcal{D} \rightarrow \mathcal{C}$ are functors and $\eta : \mathbf{1}_{\mathcal{C}} \rightarrow (G \circ F)$ and $\epsilon : (F \circ G) \rightarrow \mathbf{1}_{\mathcal{D}}$ are natural transformations between the functors such that

$$(G\epsilon) \circ (\eta G) = \mathbf{1}_G \text{ and } (\epsilon F) \circ (F\eta) = \mathbf{1}_F \quad (2.8)$$

which means that the following diagrams commute:

$$\begin{array}{ccc}
 G(B) & \xrightarrow{\eta(G(B))} & (G \circ F \circ G)(B) \\
 \searrow \text{id}_{G(B)} & & \downarrow G(\epsilon(B)) \\
 & & G(B)
 \end{array}
 \qquad
 \begin{array}{ccc}
 F(A) & & \\
 \downarrow F(\eta(A)) & \searrow \text{id}_{F(A)} & \\
 (F \circ G \circ F)(A) & \xrightarrow{\epsilon(F(A))} & F(A)
 \end{array}$$

We say that G is the right adjoint to F and that F is the left adjoint of G , and the natural transformations η and ϵ are called the two adjunctions.

A more intuitive explanation of adjointness can be found in Theorem 2.5.1 which says that when categories \mathcal{C} and \mathcal{D} are locally small categories, the adjointness (F, G, η, ϵ) can be considered as an isomorphism between the morphisms from A to $G(B)$ in \mathcal{C} and the morphisms from $F(A)$ to B in \mathcal{D} for each objects A in \mathcal{C} and B in \mathcal{D} .

Theorem 2.5.1 (Proposition 3.4. from Introduction to Higher Order Categorical Logic (Lambek J. and Scott P.J.)). *An adjointness (F, G, η, ϵ) between locally small categories (i.e. categories whose morphisms between any two objects form a set) \mathcal{C} and \mathcal{D} gives rise to and is determined by a natural isomorphism between the functors $\text{Hom}_{\mathcal{D}}(F(-), -), \text{Hom}_{\mathcal{C}}(-, G(-)) : \mathcal{C}^{op} \times \mathcal{D} \rightarrow \mathbf{Set}$, from the product of the oposite category of \mathcal{C} obtained by reversing the direction of each morphism and the category \mathcal{D} to the category of sets \mathbf{Set} .*

Adjointness is widely used to formalize various notions in category theory, for example, terminal object, which is the object T to which there is a unique morphism from each object A to T . Not every category has the terminal object, and the existence of the terminal object in category \mathcal{C} corresponds to the existence of the right adjoint of the functor $F : \mathcal{C} \rightarrow \mathcal{I}$ from the category \mathcal{C} to the category \mathcal{I} which has one object T with unique morphism id_T . Intuitive reasoning for it is that for a locally small category \mathcal{C} , the right adjoint G of the functor F defines the terminal object $G(T)$ in \mathcal{C} where there is an isomorphism between the morphisms from any object A in \mathcal{C} to $G(T)$ and the morphisms from $F(A) = T$ to T , which is to say that there is a unique morphism from A to $G(T)$ for each object A in \mathcal{C} .

Limit

Definition 2.5.2 (Definition of limit (Lambek J. and Scott P.J.)). For a given (index) category \mathcal{I} and a functor $F : \mathcal{I} \rightarrow \mathcal{C}$ (called an \mathcal{I} -diagram), a *limit* of F is given by a terminal object in the category of all pairs (A, t) with object A in \mathcal{C} and a natural transformation $t : K(A) \rightarrow F$ between the functor $K(A) : \mathcal{I} \rightarrow \mathcal{C}$ (which is a functor with constant value A) and the given functor F .

An example of the limit is the product in a category in which case the index category \mathcal{I} is discrete, meaning that there are no morphisms in \mathcal{I} except the identity morphisms. Product of a family of objects $\{A_i\}$ in \mathcal{C} is an object A called product with a family of morphisms $\{p_i : A \rightarrow A_i\}$ called projections. Moreover, product satisfies a universal property, which is that for any object B and family of morphisms $\{q_i : B \rightarrow A_i\}$, there is a unique morphism $h : B \rightarrow A$ such that $q_i = p_i \circ h$ for each i . The object A is represented as $\prod A_i$ and the unique morphism h is denoted as $\langle q_i \rangle : B \rightarrow \prod A_i$.

In terms of limit, the product of a family of objects $\{A_i\}$ is the limit of index category \mathcal{I} and functor $F : \mathcal{I} \rightarrow \mathcal{C}$ given by the discrete category with objects $\{i\}$ and the functor F which maps object i in \mathcal{I} to the object A_i in \mathcal{C} . The limit consists of an object A which is product and a natural transformation $t : K(A) \rightarrow F$ which defines the projection $p_i : A \rightarrow A_i$ for each element i in \mathcal{I} . Moreover, the universal property means that the limit (A, t) is the terminal object in the category of all pairs (A, t) .

In addition, when the family of objects is empty, the product becomes the terminal object, to which there is a unique morphism from each object. We call this object unit and denote as 1 ; and denote the unique morphism as $O_A : A \rightarrow 1$.

Monad

Definition 2.5.3 (Definition of triple (Lambek J. and Scott P.J.)). A *triple*, or a *monad*, (T, η, μ) on a category \mathcal{C} consists of a functor $T : \mathcal{C} \rightarrow \mathcal{C}$ and natural transformations $\eta : \mathbf{1}_{\mathcal{C}} \rightarrow T$ and $\mu : T^2 \rightarrow T$ satisfying the equations

$$\mu \circ T\eta = \mathbf{1}_T = \mu \circ \eta T, \quad \mu \circ \mu T = \mu \circ T\mu.$$

In other words, the following diagrams commute:

$$\begin{array}{ccc} TA & \xrightarrow{T(\eta(A))} & T^2A \\ \mu(T(A)) \downarrow & \searrow \text{id}_{TA} & \downarrow \mu(A) \\ T^2A & \xrightarrow{\mu(A)} & TA \end{array} \qquad \begin{array}{ccc} T^3A & \xrightarrow{T(\mu(A))} & T^2A \\ \mu(T(A)) \downarrow & & \downarrow \mu(A) \\ T^2A & \xrightarrow{\mu(A)} & TA \end{array}$$

for any object A in \mathcal{C} . These equations are called the *unity laws* and *associative law* respectively.

Monads are used in the formal definition of computational models and each monad produces the Kleisli category which is defined as in Definition 2.5.4.

Definition 2.5.4 (Definition of Kleisli category (Definition 2.3 in [42])). Given a monad (T, η, μ) over \mathcal{C} , the Kleisli category \mathcal{C}_T , is the category s.t. :

- the objects of \mathcal{C}_T are those of \mathcal{C}
- the set $\mathcal{C}_T(A, B)$ of morphisms from A to B in \mathcal{C}_T is $\mathcal{C}(A, TB)$

- the identity on A in \mathcal{C}_T is $A \xrightarrow{\eta_A} TA$
- the composition of $f \in \mathcal{C}_T(A, B)$ and $g \in \mathcal{C}_T(B, C)$ in \mathcal{C}_T is

$$A \xrightarrow{f} TB \xrightarrow{T(g)} T^2C \xrightarrow{\mu(C)} TC$$

2.5.2 . Higher-order

Programming languages like (simply typed) lambda-calculus allow us to define high-order functions, which refers to the functions over functions that again can be high-order functions, and to apply these functions to other terms. For example, in lambda-calculus, we can define a function by the abstraction of a term that takes variables, i.e., polynomial. The functions defined by abstraction can be applied to a term whose meaning is to substitute all variables in the polynomial by the given term.

Returning to categorical semantics, all terms are mapped to a morphism between the objects of their types in a categorical model. The abstraction necessitates a way to designate a morphism of function constructed from the morphism which denotes the polynomial, and the application requires being able to name a morphism that represents the substitution of the variables (or the replacement of the part for variable in the morphism) in the polynomial by a morphism mapped to the term.

However, before we ask these questions, we need to know if we can say that a polynomial is a morphism. A polynomial is a term dependent on an indefinite term called x . In categorical semantics, a polynomial is a family of morphisms indexed by morphism x . The problem is that we do not know how to represent a family of morphisms as a morphism in the same category. The functional completeness [36] in categorical model states that every polynomial $\phi(x) : A \rightarrow C$ in variable $x : A \rightarrow B$ is uniquely represented as $A \xrightarrow{x} B \xrightarrow{f} C$, where f is a morphism that does not depend on x .

Therefore, suppose that we let the abstraction of polynomial $\phi(x) : A \rightarrow C$ (which is a morphism $A \xrightarrow{x} B \xrightarrow{f} C$) to be the morphism $f : B \rightarrow C$ and the application $f(y)$ to be simply the composition $A \xrightarrow{y} B \xrightarrow{f} C$. Then, we can see that substitution of variable x in the polynomial $\phi(x)$ is the substitution of the morphism x by y . To sum up, the functional completeness gives us a representation of polynomial $\phi(x) : A \rightarrow C$, which is a family of morphisms indexed by the morphism $x : A \rightarrow B$ as the abstraction of the polynomial which is a morphism from B to C .

Still, we need to transform this morphism of the abstraction from B to C into a term which is a morphism from some object 1 (unit of product) to C^B an object (internal hom) connected to the type $B \rightarrow C$. Or, in more general case where there is more than one variables in the polynomial, the morphism of the polynomial is represented as $\Gamma \times A \xrightarrow{\text{id}_\Gamma \times x} \Gamma \times B \xrightarrow{f} C$ and we need to

transform the morphism $f : \Gamma \times B \rightarrow C$ into a morphism from Γ to C^B . It can be done by the adjointness structure in the cartesian closed category, and the object $B \rightarrow C$ is called internal hom or exponential object of B and C .

A standard structure for modeling higher-order programming languages is the structure of a cartesian-closed category (CCC) [36]. A CCC is a category \mathcal{C} with a binary product $(- \times -)$ and its unit 1 , and internal hom $(-)^{(-)}$ where there is an adjointness between the functors $(- \times B), (-)^B : \mathcal{C} \rightarrow \mathcal{C}$. The adjointness implies that there is an isomorphism between the sets $\text{Hom}_{\mathcal{C}}(\Gamma, A \times B)$ and $\text{Hom}_{\mathcal{C}}(\Gamma, A \rightarrow B)$ morphism when the category \mathcal{C} is a locally small category.

Type derivation rules in simply typed lambda-calculus can be obtained from the structure of CCC.

- From the definition of category:

$$\begin{aligned} \text{id}_A &: A \rightarrow A \\ g \circ f &: A \rightarrow C, \text{ for any } f : A \rightarrow B \text{ and } g : B \rightarrow C \end{aligned}$$

- From the product in category:

$$\begin{aligned} O_a &: A \rightarrow 1 \\ p_1 &: A \times B \rightarrow A \\ p_2 &: A \times B \rightarrow B \\ \langle f, g \rangle &: C \rightarrow A \times B, \text{ for any morphisms } f : C \rightarrow A \text{ and } g : C \rightarrow B \\ p_1 \circ h &: C \rightarrow A, \text{ for any morphism } h : C \rightarrow A \times B \\ p_2 \circ h &: C \rightarrow B, \text{ for any morphism } h : C \rightarrow A \times B \end{aligned}$$

The first morphism corresponds to the introduction rule of truth value T ; the second and the third morphisms represent the axiom rule; and the rest correspond to the introduction and the elimination rules of \wedge , in the intuitionistic propositional logic.

- From the adjointness $((- \times B), (-)^B, \eta, \epsilon)$:

$$\begin{aligned} \epsilon(A) &: (A^B) \times B \rightarrow A \\ \text{ev}(h_1, h_2) &: C \xrightarrow{\langle h_1, h_2 \rangle} (A^B) \times B \xrightarrow{\epsilon(A)} A, \text{ for any } h_1 : C \rightarrow A^B \text{ and } h_2 : C \rightarrow B \\ h^* &: C \xrightarrow{\eta(C)} (C \times B)^B \xrightarrow{((-)^B)(h)} A^B, \text{ for any } h : C \times B \rightarrow A \end{aligned}$$

The first morphism corresponds to the morphism called evaluation, which appears in the second morphism for the elimination rule of \rightarrow ; while the third morphism corresponds to the introduction rule of \rightarrow .

On top of the structure of CCC, we can add coproduct structure to obtain the inference rules for \vee and \perp in Definition 2.4.3.

Given that the morphism represents the proof, the commuting diagrams between morphisms create an equivalence relation among proofs. Any CCC gives the following equalities between morphisms.

- From the definition of category:

$$\begin{aligned} \text{id}_B \circ f &= f = f \circ \text{id}_A \\ h \circ (g \circ f) &= (h \circ g) \circ f \end{aligned}$$

for any morphisms $f : A \rightarrow B$, $g : B \rightarrow C$, and $h : C \rightarrow D$.

- From the product in category:

$$\begin{aligned} f &= O_A, \text{ for all } f : A \rightarrow 1 \\ p_1 \langle f, g \rangle &= f \\ p_2 \langle f, g \rangle &= g \\ \langle p_1 h, p_2 h \rangle &= h \end{aligned}$$

for any morphisms $f : C \rightarrow A$, $g : C \rightarrow B$, and $h : C \rightarrow A \times B$.

- From the cartesian closed category:

$$\begin{aligned} \epsilon(A) \langle h^* p_1, p_2 \rangle &= h \\ (\epsilon(A) \langle k p_1, p_2 \rangle)^* &= k \end{aligned}$$

for any $h : C \times B \rightarrow A$ and $k : C \rightarrow A^B$. Note that the left hand side in the first equation refers to the evaluation $\text{ev}(h^* p_1, p_2) : C \times B \rightarrow A$ where the first argument $h^* p_1$ represents the abstraction of the morphism h . Therefore, the first equation corresponds to the β -reduction in lambda-calculus applied to the abstraction h^* and the variable of type B . In other word, if is the cut-elimination in sequent calculus (meaning that the cut-rule, i.e., $C, B \vdash A$ and $B \vdash B$ imply $C, B \vdash A$, is reduced to the proof of the sequent $C, B \vdash A$, where the right branch (proof) is just an axiom rule, as shown below).

$$\frac{\frac{C, B \vdash h : A}{C \vdash h^* : A^B} \quad \frac{}{B \vdash \text{id}_B : B}}{C, B \vdash h^* p_1 : A^B} \quad \frac{}{C, B \vdash p_2 : B}}{C, B \vdash \text{ev}(h^* p_1, p_2) : B} = C, B \vdash h : A$$

Note that we put the name of the morphism corresponding to the proof between the turnstile symbol \vdash and the colon $:$. Also, note that the logic is sensitive to the change of context, which is controlled by the projection operators.

Similarly, we can show the second equation as follows. The second equation corresponds to the η -expansion which says that the term h with variable of type B is equal to the application of the abstraction h^* applied to the variable of type B .

$$\frac{\frac{C \vdash k : A^B}{C, B \vdash kp_1 : A^B} \quad \frac{\overline{B \vdash \text{id}_B : B}}{C, B \vdash p_2 : B}}{C, B \vdash \text{ev}(kp_1, p_2) : A} = C \vdash k : A^B$$

Lastly, any CCC satisfies the functional completeness property as shown in Lemma 2.5.2 from [36].

Lemma 2.5.2 (Functional completeness (Proposition 6.1. from Introduction to Higher Order Categorical Logic (Lambek J. and Scott P.J.))). *For every polynomial $\phi(x) : B \rightarrow C$ in an indeterminate $x : 1 \rightarrow A$ over a cartesian or cartesian closed category \mathcal{C} , there is a unique arrow $f : A \times B \rightarrow C$ in \mathcal{C} such that $f \langle xO_B, \text{id}_B \rangle = \phi(x)$.*

2.5.3 . Side-effects

The models described above are not enough when considering programs with side effects. Now, computation does mean not only the reduction of a program but also the transformation of the environment: let us call the mixture of the environment and the program as configuration.

A concrete example of side-effect is the quantum state. Based on the QRAM model of quantum computation [35], computation in the classical host does not only reduce the program but also modifies the quantum state through the quantum co-processor. In this case, the configuration would be represented by some structure that contains the program and the quantum state. However, we need to know how to formally define such a structure and, in particular, what structures the structure requires to have to build a consistent model of computation.

This question of categorical models of computation is studied by Eugenio Moggi [42, 43, 44]. In categorical semantics, he defines the model of computation with side-effects by the Kleisli category induced by a monad (T, η, μ) with tensorial strength on a category \mathcal{C} of the categorical model of the computation, for example, CCC for the simply-typed lambda-calculus. Roughly speaking, the functor $T : \mathcal{C} \rightarrow \mathcal{C}$ refers to the configuration space; in specific, the object $T(A)$ designates the configuration space on the object A and the morphism $T(f)$ defines a computation over the configuration space on the corresponding objects; the natural transformation $\eta : \mathbf{1}_{\mathcal{C}} \rightarrow T$, called unit, allows us to define a particular configuration on each object; and the natural transformation

$\mu : T^2 \rightarrow T$, called multiplication, states that the configuration of a configuration is a configuration and it allows us to compose programs. Moreover, the tensorial strength extends the notion of configuration to the general context represented by the product. Then, the Kleisli category induced by the monad over the category \mathcal{C} gives the model of computation where the construction of morphism is defined inductively for the program.

Formally, Definition 2.5.5, Definition 2.5.6, and Definition 2.5.7 defines the categorical model for simply typed lambda-calculus with side-effects. Assuming that the product in CCC has required properties, like symmetry and associativity, the simply-typed lambda-calculus with side-effects can be interpreted in the Kleisli category generated by the λ_c model over CCC.

Definition 2.5.5 (Definition of computational model (Definition 2.1 in [42])). A *computational model* is a monad (T, η, μ) over category \mathcal{C} which satisfies *equalizing requirement* : $\eta_A : A \rightarrow TA$ is an equalizer of η_{TA} and $T(\eta_A)$. In other word, for any $f : B \rightarrow TA$ such that $B \xrightarrow{f} TA \xrightarrow{\eta_{TA}} T^2A = B \xrightarrow{f} TA \xrightarrow{T(\eta_A)} T^2A$, there exists a unique $m : B \rightarrow A$ such that $f = B \xrightarrow{m} A \xrightarrow{\eta_A} TA$.

Definition 2.5.6 (Definition of computational cartesian model (Definition 3.2 in [42])). Let \mathcal{C} be a category with finite products equipped with the following natural isomorphisms:

$$(1 \times A) \xrightarrow{r} A, (A \times B) \times C \xrightarrow{\alpha} A \times (B \times C), (A \times B) \xrightarrow{\sigma} (B \times A)$$

Then, a *computational cartesian model* over \mathcal{C} is a computational model (T, η, μ) over \mathcal{C} together with a tensorial strength $t_{A,B} : (A \times TB) \rightarrow T(A \times B)$ of T , which is a natural transformation satisfying the following diagrams:

$$\begin{array}{ccc}
1 \times TA & \xrightarrow{t_{1,A}} & T(1 \times A) \\
& \searrow^{r_{TA}} & \downarrow T_{r_A} \\
& & TA
\end{array}$$

$$\begin{array}{ccc}
(A \times B) \times TC & \xrightarrow{t_{A \times B, C}} & T((A \times B) \times C) \\
\downarrow \alpha_{A,B,TC} & & \downarrow T(\alpha_{A,B,C}) \\
A \times (B \times TC) & \xrightarrow{\text{id}_A \times t_{B,C}} & A \times T(B \times C) \xrightarrow{t_{A, B \times C}} & T(A \times (B \times C))
\end{array}$$

$$\begin{array}{ccc}
A \times B & \xrightarrow{\text{id}_{A \times B}} & A \times B \\
\downarrow \text{id}_A \otimes \eta(B) & & \downarrow \eta(A \times B) \\
A \times TB & \xrightarrow{t_{A,B}} & T(A \times B) \\
\uparrow \text{id}_B \times \mu(B) & & \uparrow \mu(A \times B) \\
A \times T^2B & \xrightarrow{t_{A', TB}} & T(A \times TB) \xrightarrow{T(t_{A,B})} & T^2(A \times B)
\end{array}$$

Definition 2.5.7 (Definition of λ_c -model over \mathcal{C} (Definition 3.5 in [42])). Let \mathcal{C} be a category with finite products. A λ_c -model over \mathcal{C} is a computational cartesian model (T, η, μ, t) over \mathcal{C} together with a family of universal arrows $\text{eval}_{A,B}^T : (B_T^A \times A) \rightarrow TB$ in \mathcal{C} s.t. for any $f : (C \times A) \rightarrow TB$, there exists a unique $h : C \rightarrow B_T^A$ (denoted by $\Lambda_{A,B,C}^T(f)$) making the following diagram commute.

$$\begin{array}{ccc} B_T^A \times A & \xrightarrow{\text{eval}_{A,B}^T} & TB \\ h \times \text{id}_A \uparrow & \nearrow f & \\ C \times A & & \end{array}$$

Finally, to make a formal model of computation with side-effects, it only remains to define the side-effects in terms of the computational model in Definition 2.5.5. Some examples of computational models are non-deterministic computation and computations with side-effects:

- non-deterministic computation (over the category **Set**):

The functor $T(-)$ is the powerset functor which sends an object A to its powerset $\mathcal{P}(A)$ and a morphism $f : A \rightarrow B$ to the function from $\mathcal{P}(A)$ to $\mathcal{P}(B)$ which maps each subset of A to the image $f(A)$ of A . Next, the unit $\eta(A)$ maps each element $a \in A$ to the singleton set $\{a\}$ and the multiplication $\mu(A)$ is the set-theoretic union.

- computation with side-effects:

The monad is given by: the functor $T(-) = (- \times S)^S$ which maps an object A to the object $(A \times S)^S$ whose intuitive meaning is a function dependent on the memory S that returns a value and updates the memory; the unit $\eta(A)$ is the map $a \mapsto (\lambda s : S. \langle a, s \rangle)$; and the multiplication $\mu(A)$ is the map $f \mapsto (\lambda s : S. \text{eval}(fs))$.

For the example of quantum computation introduced before, a quantum state in the quantum co-processor can be represented as a complex function over the computational basis. Therefore, in terms of monads, quantum computation can be modeled by the computation with side-effects or by the distribution monad.

2.5.4 . Properties of categorical semantics

Categorical semantics allows us to consider a term in terms of a mathematical object instead of the possibly infinite sequence of reductions of the term or all possible effects of the term in any context. However, what it means to have a categorical semantics depends on the properties that the semantics satisfies. This section introduces four properties of categorical semantics that state the relationship between the models in the categorical semantics and the equivalence classes of terms.

Equivalence of terms

The notion of equivalence of terms can vary across the context, but there is an exemplary notion of equivalence, namely, observational equivalence. First, the equivalence is based on an operational semantics which consists of a set of reduction rules over configurations. Two terms are equivalent whenever they both diverge or reduce to the same value. However, when a term has side-effects, a term may have different behaviors depending on the context. In this case, the equivalence of two terms means that the terms are equivalent in any context; in other words, we can substitute one term with the other in any context.

More concretely, for example, we can define the context $C[-]$ (at the level of term) and $\mathbf{C}[-]$ (at the level of configurations) over some language \mathcal{L} , as shown in Eq. (2.9).

$$\begin{aligned}
C[-] ::= & [-] \mid \lambda x.C[-] \mid M(C[-]) \mid (C[-])M \mid \langle C[-], M \rangle \mid \langle M, C[-] \rangle \\
& \mid \text{let } \langle x, y \rangle = C[-] \text{ in } M \mid \text{let } \langle x, y \rangle = M \text{ in } C[-] \\
& \mid \text{if } C[-] \text{ then } M_a \text{ else } M_b \\
& \mid \text{if } M \text{ then } C[-] \text{ else } M_b \mid \text{if } M \text{ then } M_a \text{ else } C[-] \\
\mathbf{C}[-] ::= & (Q, C[-])
\end{aligned} \tag{2.9}$$

for any term M , M_a , and M_b in the language \mathcal{L} and any context $Q \in C[-]$ which means state. Then, observational equivalence of two terms are defined in Definition 2.5.8.

Definition 2.5.8. Two term M_1 and M_2 are observationally equivalent if, for any configuration context $\mathbf{C}[-]$, either there exists a configuration \mathbf{C}_t such that

$$\mathbf{C}[M_1] \rightarrow^* \mathbf{C}_t \text{ and } \mathbf{C}[M_2] \rightarrow^* \mathbf{C}_t$$

or both configurations diverge.

Properties of categorical semantics

Next, we introduce the following properties of categorical semantics.

- Soundness

On the basis, soundness means that the denotation of the term is preserved by the equivalence of terms. On the one hand, when the computation is pure function, we can define the equivalence of terms by the reduction rules from operational semantics, i.e., two terms are equivalent whenever they reduce to the same value or both diverge. On the other hand, when the computation has side-effects, we can define an equivalence on terms as the observational equivalence. Regardless of the definition of an equivalence class, we call this property of categorical semantics soundness defined as in Definition 2.5.9 which depends on

the notion of equivalence. Note that $\llbracket - \rrbracket$ refers to the morphism in the categorical model constructed according to the interpretation rules for each typing derivation.

Definition 2.5.9. For any typing context Γ and state Q and type A , if two terms M_1 and M_2 are equivalent under the context $\Gamma \vdash (Q, -) : A$, then

$$\llbracket \tau_1 \mid \Gamma \vdash (Q, M_1) : A \rrbracket = \llbracket \tau_2 \mid \Gamma \vdash (Q, M_2) : A \rrbracket$$

for any typing derivation τ_1 of $\Gamma \vdash (Q, M_1) : A$ and τ_2 of $\Gamma \vdash (Q, M_2) : A$.

- Full completeness

After showing the soundness property of categorical semantics, we may want to prove that this model precisely represents the meaning of terms. To explain, it reduces to showing the one-to-one correspondence between all possible denotations in the model and all equivalence classes of terms. Again the equivalence class can be defined by either the reduction rules or observational equivalence. Showing this one-to-one correspondence corresponds to showing full completeness of the categorical model depending on the choice of the notion of equivalence of terms. This property can be defined as in Definition 2.5.10.

Definition 2.5.10. Two terms M_1 and M_2 are equivalent under the context $\Gamma \vdash (Q, -) : A$ if and only if

$$\llbracket \tau_1 \mid \Gamma \vdash (Q, M_1) : A \rrbracket = \llbracket \tau_2 \mid \Gamma \vdash (Q, M_2) : A \rrbracket$$

for any typing derivation τ_1 of $\Gamma \vdash (Q, M_1) : A$ and τ_2 of $\Gamma \vdash (Q, M_2) : A$.

Moreover, for each model \mathcal{M} in the semantics, there exists a typing derivation τ of $! \Delta \vdash M : A$ whose denotation $\llbracket \tau \mid \Gamma \vdash (Q, M) : A \rrbracket$ is \mathcal{M} .

- Full abstraction

There are some cases where we do not need the full completeness of categorical semantics, although we want to show that it represents the precise meaning of terms. It corresponds to the full abstraction (Definition 2.5.11) where there can be models in the semantics which are not a denotation of any term. Still, full abstraction requires the semantics to be capable of representing the precise meaning of the term, which means that there is a one-to-one correspondence between a subset of all models and all equivalence classes of terms, i.e., an injection from the equivalence classes of terms to the models.

Definition 2.5.11. Two terms M_1 and M_2 are equivalent under the context $\Gamma \vdash (Q, -) : A$ if and only if

$$\llbracket \tau_1 \mid \Gamma \vdash (Q, M_1) : A \rrbracket = \llbracket \tau_2 \mid \Gamma \vdash (Q, M_2) : A \rrbracket$$

for any typing derivation τ_1 of $\Gamma \vdash (Q, M_1) : A$ and τ_2 of $\Gamma \vdash (Q, M_2) : A$.

- Adequacy

Another relaxation of the property of categorical semantics from full abstraction is adequacy. Adequacy requires that the equality of the denotations of two terms implies their equivalence. Note that it does not imply that all terms in an equivalence class are mapped to the same model in the semantics, and, hence, it does not imply the soundness property. In effect, adequacy is the converse of the soundness theorem, and together with soundness, adequacy implies full abstraction. Still, adequacy is a beneficial property of categorical semantics, which allows us to show the equality of terms by comparing the semantics without reducing or executing the terms. Adequacy property can be represented as in Definition 2.5.12.

Definition 2.5.12. For any two terms M_1 and M_2 ; any typing context Γ ; any state Q ; any type A ; and typing derivations τ_1 and τ_2 of the type judgements $\Gamma \vdash (Q, M_1) : A$ and $\Gamma \vdash (Q, M_2) : A$: if

$$\llbracket \tau_1 \mid \Gamma \vdash (Q, M_1) : A \rrbracket = \llbracket \tau_2 \mid \Gamma \vdash (Q, M_2) : A \rrbracket$$

then, the terms M_1 and M_2 are equivalent.

2.6 . Linear logic

Linear logic, introduced by Girard [25], is a logical system where one considers a proof as a resource. Linear logic is a resource-sensitive system that incorporates both intuitionistic and classical logic. There are multiple variants [73, 26, 22] of linear logic, which contains different sets of connectives, units, implications, exponentials, and quantifiers.

For the sake of quantum programming languages, we only introduce an intuitionistic propositional fragment of linear logic that contains \multimap (multiplicative conjunctive connective), I (multiplicative conjunctive unit), \multimap (linear implication), $!$ (exponential), and variables. Simply, the language can be represented as a propositional logic $(I, !, \otimes, \multimap)$ with one constant I , one unary connective $!$ and two binary connectives \otimes and \multimap . Formally, the language of the logic is defined inductively as in Definition 2.6.1.

Definition 2.6.1. Language of intuitionistic, multiplicative propositional linear logic with exponential is defined as follows.

$$S, S_1, S_2 ::= I \mid v \mid S_1 \otimes S_2 \mid S_1 \multimap S_2 \mid !S$$

where v refers to variables.

The unit I of the tensor can be considered as linear true and tautologies, which are properties logically derived from true, are the properties that follow from I . The multiplicative conjunction $S_1 \otimes S_2$ represents a juxtaposition of propositions whose derivation is also a juxtaposition of the derivations of the two propositions. The linear arrow $S_1 \multimap S_2$ is another connective representing a juxtaposition of propositions whose derivation requires the derivation of S_2 given the derivation of S_1 . Lastly, exponential $!S$ represents a proposition whose derivation is treated classically, meaning it can be duplicated and erased.

The logic comes with inference rules which conform to the explanation given above. We introduce the inference rules in the form of sequent calculus as usual. A sequent has the form $A_1, \dots, A_n \vdash B$ which consists of a set of propositions A_i , for $1 \leq i \leq n$, and a proposition B . A sequent is read as one can construct a proof for proposition B given proofs of propositions $(A_i)_{1 \leq i \leq n}$. How we construct such a proof is described by a set of inference rules, as in definition 2.6.2, which forms a derivation tree whose nodes correspond to sequents and root is the conclusion.

Definition 2.6.2. The following set of inference rules corresponds to the intuitionistic, multiplicative propositional linear logic. Each inference rule is read as: given the derivation of the sequents above the horizontal line, we can construct a derivation of the sequent under the line.

$$\begin{array}{c} \frac{}{A \vdash A} (\text{HYP}) \quad \frac{}{\vdash I} (I_I) \quad \frac{\Delta_1 \vdash I \quad \Delta_2 \vdash A}{\Delta_1, \Delta_2 \vdash A} (I_E) \\ \\ \frac{\Delta_1 \vdash A_1 \quad \Delta_2 \vdash A_2}{\Delta_1, \Delta_2 \vdash A_1 \otimes A_2} (\otimes_I) \quad \frac{\Delta_1 \vdash A_1 \otimes A_2 \quad \Delta_2, A_1, A_2 \vdash A}{\Delta_1, \Delta_2 \vdash A} (\otimes_E) \\ \\ \frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B} (\multimap_I) \quad \frac{\Delta_1 \vdash A \multimap B \quad \Delta_2 \vdash A}{\Delta_1, \Delta_2 \vdash B} (\multimap_E) \\ \\ \frac{\Delta \vdash A \quad \forall A \in \Delta. (\exists A'. (A = !A'))}{\Delta \vdash !A} (!_I) \quad \frac{\Delta_1 \vdash !A \quad A, \Delta_2 \vdash B}{\Delta_1, \Delta_2 \vdash B} (!_E) \\ \\ \frac{\Delta, \vdash A}{\Delta, !B \vdash !A} (!_W) \quad \frac{\Delta, !B, !B \vdash A}{\Delta, !B \vdash !A} (!_C) \end{array}$$

Note that the symbols Δ , Δ_1 and Δ_2 refer to the set of propositions.

2.7 . Semantics of linear logic

Linear logic appears in the type systems of various quantum programming languages, including quantum lambda-calculus and quantum circuit languages like QWire and Quipper. In particular, quantum data is typed with a linear quantum type, called a qubit, and with the tensor product of the quantum types, while classical data has a non-linear type which can be utilized any number of times in the term and type. Consequently, the denotational semantics of a quantum programming language and its type system rely on the semantics of linear logic. This section summarizes the categorical semantics of the intuitionistic propositional and multiplicative part of linear logic with exponential.

Our presentation relies on the presentation from the thesis of Benoît Valiron [76] (Chapter 2.7 for the monoidal category and Chapter 5.6 for the semantics of linear logic). We, first, introduce symmetric monoidal categories as categorical semantics of the multiplicative fragment of intuitionistic linear logic. Next, we define categorical structures—commutative comonoid, comonad, and its coalgebra, monoidal functor and natural transformation, and monoidal comonad—which are used to define linear and non-linear categories [14, 13, 10, 9, 11]. When discussing the categorical semantics of linear logic, we maintain our strategy of interpreting types as objects and the inference rules as constructions of morphisms in the category.

In the previous section, we have introduced linear logic where the exponential represents non-linear types. We can divide the logic into the purely linear part—all types are linear—and the classical part—types can contain exponential constructors. To explain, in purely linear logic, all propositions in the context are used precisely once in the type derivation since each leaf of the typing derivation will be either (HYP) or (I_I) and each occurrence of a proposition in the conclusion appears in exactly one leaf. In terms of categorical semantics, the purely linear part of linear logic corresponds to symmetric monoidal closed categories.

A (symmetric) monoidal category is a category with an operator of composition (\otimes) of objects, which is (symmetric), associative, and has a unit (Definition 2.7.1). Moreover, when there is an exponential object $(-)^{(-)}$ and an adjunction between the functor $(- \otimes B)$ and $(-)^B$, it is called a (symmetric) monoidal closed category, where the adjunction provides the interpretation of abstraction and application of linear function type constructors \multimap .

Definition 2.7.1 (Monoidal category and symmetric monoidal category (Definition 2.7.1. and Definition 2.7.2. from [76])). A monoidal category \mathcal{C} is a tuple $(\mathcal{C}, \otimes, I, \alpha, \lambda, \rho)$ where \mathcal{C} is a category, $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ is a functor, I is an object

of \mathcal{C} and α, λ, ρ are three natural isomorphisms:

$$\begin{aligned}\alpha_{A,B,C} &: A \otimes (B \otimes C) \rightarrow (A \otimes B) \otimes C \\ \lambda_A &: I \otimes A \rightarrow A \\ \rho_A &: A \otimes I \rightarrow A\end{aligned}$$

such that the diagrams

$$\begin{array}{ccc} A \otimes (B \otimes (C \otimes D)) & \xrightarrow{\alpha} & (A \otimes B) \otimes (C \otimes D) \\ \downarrow \text{id} \otimes \alpha & & \downarrow \alpha \\ A \otimes ((B \otimes C) \otimes D) & \xrightarrow{\alpha} (A \otimes (B \otimes C)) \otimes D \xrightarrow{\alpha \otimes \text{id}} & ((A \otimes B) \otimes C) \otimes D \end{array}$$

$$\begin{array}{ccc} A \otimes (I \otimes C) & \xrightarrow{\alpha} & (A \otimes I) \otimes C \\ \searrow \text{id} \otimes \lambda & & \swarrow \rho \otimes \text{id} \\ & A \otimes C & \end{array} \quad \begin{array}{c} I \otimes I \\ \rho \left(\downarrow \right) \lambda \\ I \end{array}$$

commute. We call the category *strict monoidal* when $\alpha_{A,B,C}$, λ_A and ρ_A are identity morphisms. A monoidal category is said to be *symmetric* when it is equipped with a natural isomorphism $\sigma_{A,B} : A \otimes B \rightarrow B \otimes A$ such that the diagrams

$$\begin{array}{ccc} A \otimes B & \xrightarrow{\sigma_{A,B}} & B \otimes A \\ & \searrow \sigma_{B,A} & \swarrow \\ & & \end{array} \quad \begin{array}{ccc} B \otimes I & \xrightarrow{\sigma_{B,I}} & I \otimes B \\ & \searrow \rho_B & \downarrow \lambda_B \\ & & B \end{array}$$

$$\begin{array}{ccc} A \otimes (B \otimes C) & \xrightarrow{\alpha} & (A \otimes B) \otimes C \xrightarrow{\sigma} & C \otimes (A \otimes B) \\ \downarrow \text{id} \otimes \sigma & & & \downarrow \alpha \\ A \otimes (C \otimes B) & \xrightarrow{\alpha} & (A \otimes C) \otimes B \xrightarrow{\sigma \otimes \text{id}} & (C \otimes A) \otimes B \end{array}$$

commute.

There are various contexts where symmetric monoidal categories can help to capture the structure of the subject of interest. We take two possible perspectives on the structure of symmetric monoidal category as examples. On the one hand, each object of this category may be considered a space of vectors and each morphism may be considered as linear maps as in the category of finite dimensional vector spaces. On the other hand, morphisms in a (symmetric) monoidal category can be viewed as diagrams where an object is represented as a wire, and a morphism creates a box from wires to wires [61]—note that the composition of objects is represented as the juxtaposition of wires in the diagram. The diagrammatic representation gives us a clear intuition of why a symmetric monoidal category models the purely linear part of linear logic since each wire appears precisely once in the corresponding diagram.

On top of the purely linear part, there is a non-linear part of the logic. First of all, to interpret non-linear types in category theory, we need objects with an extra structure for duplication and deletion (commutative comonoid 2.7.2 subject to different coherence conditions). Subcategories obtained by taking these particular objects from the original category and all morphisms between them form cartesian closed categories corresponding to intuitionistic logic.

Secondly, for the inference rules for introduction and elimination of $!$ ($!_I$ and $!_E$), we need to be able to lift a linear object to a non-linear object and force a non-linear object to a linear object. In categorical semantics, it is done by a comonad 2.7.3. This comonad and the comonoid structure form the basis of Bierman's linear categories [14, 13] which is formally defined below.

Definition 2.7.2 (Commutative comonoid (Definition 2.7.3. from [76])). In a symmetric monoidal category \mathcal{C} , a *commutative comonoid object* is an object A of \mathcal{C} equipped with two arrows $\diamond_A : A \rightarrow I$ and $\Delta_A : A \rightarrow A \otimes A$ such that the following diagrams commute:

$$\begin{array}{ccc}
 & A & \\
 \Delta_A \swarrow & & \searrow \Delta_A \\
 A \otimes A & & A \otimes A \\
 \downarrow A \otimes \Delta_A & & \downarrow \Delta_A \otimes A \\
 A \otimes (A \otimes A) & \xrightarrow{\alpha} & (A \otimes A) \otimes A
 \end{array}$$

$$\begin{array}{ccc}
 A \otimes I & \xleftarrow{A \otimes \diamond_A} & A \otimes A & \xrightarrow{\diamond_A \otimes A} & I \otimes A \\
 \rho_A \downarrow & & & & \downarrow \lambda_A \\
 A & \xlongequal{\quad} & A & \xlongequal{\quad} & A
 \end{array}$$

$$\begin{array}{ccc}
 & A & \\
 \Delta_A \swarrow & & \searrow \Delta_A \\
 A \otimes A & \xrightarrow{\sigma_{A,A}} & A \otimes A
 \end{array}$$

A morphism $f : (A, \diamond_A, \Delta_A) \rightarrow (B, \diamond_B, \Delta_B)$ of commutative comonoids is an arrow $f : A \rightarrow B$ in \mathcal{C} such that the following diagrams commute.

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 \downarrow \Delta_A & & \downarrow \Delta_B \\
 A \otimes A & \xrightarrow{f \otimes f} & B \otimes B
 \end{array}$$

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \\
 \searrow \diamond_A & & \swarrow \diamond_B \\
 & I &
 \end{array}$$

Definition 2.7.3 (Definition of comonad (Definition 2.6.1. from [76])). A *comonad* in a category \mathcal{C} is a tuple (L, ϵ, δ) where $L : \mathcal{C} \rightarrow \mathcal{C}$ is a functor and $\epsilon : L \rightarrow \mathbf{1}_{\mathcal{C}}$ and $\delta : L \rightarrow L^2$ are natural transformations which render commutative the diagrams

$$\begin{array}{ccc}
 LA & \xrightarrow{\delta_A} & L^2 A \\
 \delta_A \downarrow & & \downarrow L\delta_A \\
 L^2 A & \xrightarrow{\delta_{LA}} & L^3 A
 \end{array}$$

$$\begin{array}{ccccc}
 & LA & & & \\
 \text{id}_{LA} \swarrow & & \downarrow \delta_A & & \searrow \text{id}_{LA} \\
 LA & \xleftarrow{\epsilon_{LA}} & L^2 A & \xrightarrow{L\epsilon_A} & LA
 \end{array}$$

Definition 2.7.4 (Definition of L -coalgebra (Definition 2.6.3. from [76])). Given a comonad (L, δ, ϵ) on a category \mathcal{C} , an L -coalgebra is a pair $(A, h_A : A \rightarrow LA)$ where A is an object and h_A is a morphism of \mathcal{C} such that both commute.

$$\begin{array}{ccc} A & \xrightarrow{h_A} & LA \\ \downarrow h_A & & \downarrow \delta_A \\ LA & \xrightarrow{Lh_A} & L^2A \end{array} \quad \begin{array}{ccc} A & \xrightarrow{h_A} & LA \\ \searrow \text{id}_A & & \swarrow \epsilon_A \\ & A & \end{array}$$

A morphism $f : (A, h_A) \rightarrow (B, h_B)$ of L -coalgebras is a morphism $f : A \rightarrow B$ of \mathcal{C} which renders commutative the diagram

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow h_A & & \downarrow h_B \\ LA & \xrightarrow{Lf} & LB \end{array}$$

Definition 2.7.5 (Lax (symmetric) monoidal functor (Definition 2.8.1. from [76])). A (lax) monoidal functor F between two monoidal categories $(\mathcal{C}, \otimes, I)$ and $(\mathcal{D}, \otimes', I')$ consists of a functor $F : \mathcal{C} \rightarrow \mathcal{D}$ together with two natural transformations

$$d_{A,B}^F : FA \otimes' FB \rightarrow F(A \otimes B), \quad d_I^F : I' \rightarrow F(I),$$

called *coherence maps*, with the following coherence equations:

$$\begin{array}{ccc} (FA \otimes' FB) \otimes' FC & \xrightarrow{\alpha} & FA \otimes' (FB \otimes' FC) \\ \downarrow d_{A,B \otimes' C}^F & & \downarrow FA \otimes' d_{B,C}^F \\ F(A \otimes B) \otimes' FC & & FA \otimes' F(B \otimes C) \\ \downarrow d_{A \otimes B, C}^F & & \downarrow d_{A, B \otimes C}^F \\ F((A \otimes B) \otimes C) & \xrightarrow{F\alpha} & F(A \otimes (B \otimes C)) \end{array}$$

$$\begin{array}{ccc} FA \otimes' I' & \xrightarrow{FA \otimes' d} & FA \otimes' F(I) \\ \downarrow \rho & & \downarrow d_{A, I'}^F \\ FA & \xleftarrow{F\rho} & F(A \otimes I) \end{array} \quad \begin{array}{ccc} I' \otimes' FB & \xrightarrow{d \otimes' FB} & F(I) \otimes' FB \\ \downarrow \lambda & & \downarrow d_{I', B}^F \\ FB & \xleftarrow{F\lambda} & F(I \otimes B) \end{array}$$

If the categories are symmetric, the functor is said to be *lax symmetric monoidal* if the following diagram also commutes:

$$\begin{array}{ccc} FA \otimes' FB & \xrightarrow{\sigma} & FB \otimes' FA \\ \downarrow d_{A,B}^F & & \downarrow d_{B,A}^F \\ F(A \otimes B) & \xrightarrow{F\sigma} & F(B \otimes A) \end{array}$$

Definition 2.7.6 (Monoidal natural transformation (Definition 2.8.4. from [76])). If (F, d) and (F', d') are two symmetric monoidal functors from $(\mathcal{C}, \otimes, I)$ to $(\mathcal{D}, \otimes', I')$, we say that a natural transformation $n : (F, d) \rightarrow (F', d')$ is *monoidal* if the following diagrams commute:

$$\begin{array}{ccc} FA \otimes' FB & \xrightarrow{n_A \otimes' n_B} & F'A \otimes' F'B \\ \downarrow d_{A,B} & & \downarrow d'_{A,B} \\ F(A \otimes B) & \xrightarrow{n_{A \otimes B}} & F'(A \otimes B) \end{array} \quad \begin{array}{ccc} & I' & \\ d_I \swarrow & & \searrow d'_I \\ F(I) & \xrightarrow{n_I} & F'(I) \end{array}$$

Definition 2.7.7 ((Symmetric) monoidal comonad (Definition 2.8.6. from [76])). Let $(\mathcal{C}, \otimes, I)$ be a (symmetric) monoidal category. A (symmetric) *monoidal comonad* on \mathcal{C} is a comonad (L, δ, ϵ) :

- equipped with two natural transformations $d_{A,B} : LA \otimes LB \rightarrow L(A \otimes B)$ and $d_I : L(I) \rightarrow I$ making (L, d) a lax (symmetric) monoidal functor,
- such that δ and ϵ are monoidal natural transformations, i.e. such that the following diagrams commute:

$$\begin{array}{ccc} LA \otimes LB & \xrightarrow{d_{A,B}} & L(A \otimes B) \\ & \searrow \epsilon_A \otimes \epsilon_B & \swarrow \epsilon_{A \otimes B} \\ & & A \otimes B \end{array} \quad \begin{array}{ccc} I & \xrightarrow{d_I} & L(I) \\ & \searrow \text{id}_I & \swarrow \epsilon_I \\ & & I \end{array}$$

$$\begin{array}{ccc} LA \otimes LB & \xrightarrow{d_{A,B}} & L(A \otimes B) \\ \downarrow \delta_A \otimes \delta_B & & \downarrow \delta_{A \otimes B} \\ L^2A \otimes L^2B & \xrightarrow{d_{L^2A, L^2B}} & L(LA \otimes LB) \xrightarrow{Ld_{A,B}} L^2(A \otimes B) \end{array} \quad \begin{array}{ccc} I & \xrightarrow{d_I} & L(I) \\ \downarrow d_I & & \downarrow \delta_I \\ L(I) & \xrightarrow{Ld_I} & L^2(I) \end{array}$$

Definition 2.7.8 (Bierman's linear category (Definition 5.6.1. and Definition 5.6.2. from [76])). Let $(\mathcal{C}, \otimes, I, \alpha, \lambda, \rho, \sigma)$ be a symmetric monoidal category. Let $(L, \delta, \epsilon, d^L, d_I^L)$ be a monoidal comonad. We say that L is a *linear exponential comonad* provided that

1. each object in \mathcal{C} of the form LA is equipped with a commutative comonoid $(LA, \Delta_A, \Diamond_A)$ where $\Delta_A : LA \rightarrow LA \otimes LA$ and $\Diamond_A : LA \rightarrow I$.
2. Δ_A and \Diamond_A are monoidal natural transformations, i.e., the following diagrams commute.

$$\begin{array}{ccc} LA \otimes LB & \xrightarrow{\Delta_A \otimes \Delta_B} & (LA \otimes LA) \otimes (LB \otimes LB) \\ \downarrow d_{A,B}^L & & \downarrow sw \\ & & (LA \otimes LB) \otimes (LA \otimes LB) \\ & & \downarrow d_{A,B}^L \otimes d_{A,B}^L \\ L(A \otimes B) & \xrightarrow{\Delta_{A \otimes B}} & L(A \otimes B) \otimes L(A \otimes B) \end{array}$$

$$\begin{array}{ccc}
I & \xrightarrow{\lambda_I^{-1}} & I \otimes I \\
\downarrow d_I^L & & \downarrow d_I^L \otimes d_I^L \\
L(I) & \xrightarrow{\Delta_I} & L(I) \otimes L(I)
\end{array}
\quad
\begin{array}{ccc}
LA \otimes LB & \xrightarrow{\diamond_A \otimes \diamond_B} & I \otimes I \\
\downarrow d_{A,B}^L & & \downarrow \lambda_I \\
L(A \otimes B) & \xrightarrow{\diamond_{A \otimes B}} & I
\end{array}$$

$$\begin{array}{ccc}
I & \xrightarrow{\text{id}} & I \\
\searrow d_I^L & & \swarrow \diamond_I \\
& & L(I)
\end{array}$$

where sw is the natural transformation $\alpha; (\alpha^{-1} \otimes \text{id}); ((\text{id} \otimes \sigma) \otimes \text{id}); (\alpha \otimes \text{id}); \alpha^{-1}$.

3. The maps

$$\begin{aligned}
\Delta_A &: (LA, \delta_A) \rightarrow (LA \otimes LA, (\delta_A \otimes \delta_A); d_A), \\
\diamond_A &: (LA, \delta_A) \rightarrow (I, d_I^L)
\end{aligned}$$

are L -coalgebra morphisms, i.e., the following diagrams commute.

$$\begin{array}{ccc}
LA & \xrightarrow{\Delta_A} & LA \otimes LA \\
\downarrow \delta_A & & \downarrow \delta_A \otimes \delta_A \\
L^2 A & \xrightarrow{L\Delta_A} & L(LA \otimes LA) \\
& & \downarrow d_{LA, LA}^L
\end{array}
\quad
\begin{array}{ccc}
LA & \xrightarrow{\diamond_A} & I \\
\downarrow \delta_A & & \downarrow d_I^L \\
L^2 A & \xrightarrow{L\diamond_A} & L(I)
\end{array}$$

4. Every map δ_A is a comonoid morphism $(LA, \diamond_A, \Delta_A) \rightarrow (L^2 A, \diamond_{LA}, \Delta_{LA})$, i.e. the following diagrams commute.

$$\begin{array}{ccc}
LA & \xrightarrow{\delta_A} & L^2 A \\
\downarrow \Delta_A & & \downarrow \Delta_{LA} \\
LA \otimes LA & \xrightarrow{\delta_A \otimes \delta_A} & L^2 A \otimes L^2 A
\end{array}
\quad
\begin{array}{ccc}
LA & \xrightarrow{\delta_A} & L^2 A \\
\searrow \diamond_A & & \swarrow \diamond_{LA} \\
& & I
\end{array}$$

A *linear category* is a symmetric monoidal category $(\mathcal{C}, \otimes, I)$ with finite product $(\times, 1)$, together with a linear exponential comonad L .

Benton's linear-non-linear categories [10, 9, 11] in Definition 2.7.9 formulate the above categories in terms of adjunction [76]. In particular, the comonad and the comonoid structure of Bierman's linear category arise from the symmetric monoidal adjunction between the symmetric monoidal closed category and the cartesian category. The adjunction provides a comonad $(!, \epsilon :! \Rightarrow \text{Id}_L, \delta :! \Rightarrow !!)$ with the functor $! := F \circ G : \mathcal{C} \rightarrow \mathcal{C}$ and the natural transformations: ϵ , the counit of the adjunction, and δ , defined by $\delta(A) := F(\eta_G(A))$ where $\eta : \text{Id}_C \Rightarrow F \circ G$ is the unit of the adjunction.

Definition 2.7.9 (Benton’s linear-non-linear category (Definition 5.6.3. from [76])). A linear/non-linear category consists of

- a symmetric monoidal closed category with finite coproducts (L, \otimes, I) ;
- a category $(\mathcal{C}, \times, 1)$ with finite products;
- a symmetric monoidal adjunction

$$\begin{array}{ccc} & F & \\ & \curvearrowright & \\ C & & L \\ & \curvearrowleft & \\ & G & \\ & \perp & \end{array}$$

2.8 . Lambda-Calculi with Linear Type

Linear lambda-calculus is a programming language that reflects the purely linear part of linear logic. The types consist of the language (propositions) of the intuitionistic propositional linear logic without exponential $!$. The types also include constants types. The terms represent the type inference rules and the derivation trees. Linear lambda-calculus is used for quantum lambda calculus, where all variables, classical or quantum, are used precisely once. An example of quantum lambda-calculus with linear type is shown in Definition 2.8.1 which is taken from the thesis of Benoît Valiron [76] (Chapter 7).

Definition 2.8.1 (Linear quantum lambda calculus (Definition 7.1.1. from [76])).

$$\begin{aligned} (\text{Type}) \quad A, B & ::= \text{bit} \mid \text{qubit} \mid A \multimap B \mid 1 \mid A \otimes B \\ (\text{Term}) \quad M, N, P & ::= x^A \mid MN \mid \lambda x^A.M \mid \langle M, N \rangle \mid \text{let } \langle x^A, y^B \rangle = M \text{ in } N \\ & \mid \text{if } M \text{ then } N \text{ else } P \mid 0 \mid 1 \mid \text{new} \mid \text{meas} \mid U \\ & \mid \text{let } * = M \text{ in } N \mid * \mid \Omega_{x_1^{A_1}, \dots, x_n^{A_n}}^A \end{aligned}$$

where Ω is a non-terminating term and $\text{let } * = M \text{ in } N$ is used to match 0-tuples.

The quantum related constant added to types is qubit for qubit and constants added to terms are new for initialization, meas for measurement, and U for unitary maps. The term constants have the following types:

$$\text{meas} : \text{qubit} \multimap \text{bit}, \quad \text{new} : \text{bit} \multimap \text{qubit}, \quad U : \text{qubit}^{\otimes n} \multimap \text{qubit}^{\otimes n}$$

for some integer n depending on the unitary map U .

The operational semantics is defined over the quantum closure which is triple $[Q, L, M]$ where Q and L represent the quantum state (a normalized vector in $\otimes_{i=1}^n \mathbb{C}^2$) and a bijective function from the variables of the term M

to $\{0, \dots, n-1\}$ for some $n \geq 0$. The quantum state Q and the bijection L represent the quantum memory accessed through the quantum co-processor. The quantum related terms (new, meas, and U) only modify the quantum memory.

For the denotational semantics, an instance of the model of the purely linear logic (symmetric monoidal closed category) called **CPM** (Definition 2.8.5) is used for a fully abstract model. This category has a product \otimes and a coproduct \oplus , and the function type $A \multimap B$ is interpreted as the product $\llbracket A \rrbracket \otimes \llbracket B \rrbracket$, which induces the adjunction given by the associativity of the product.

Definition 2.8.2 (Positive maps from [46]). A matrix $A \in \mathbb{C}^{n \times n}$ is positive if $v^* A v > 0$ for all $v \in \mathbb{C}^n$.

Definition 2.8.3 (Löwner partial order from [46]). Given $A, B \in \mathbb{C}^{n \times n}$, we write $A \sqsubseteq B$ iff $B - A$ is positive and it is called Löwner partial order.

Definition 2.8.4 (Completely positive maps from [46]). A linear map $F : \mathbb{C}^{n \times n} \rightarrow \mathbb{C}^{m \times m}$ is called positive if $A \sqsupseteq 0$ implies $F(A) \sqsupseteq 0$, and completely positive if $F \otimes \text{id}_k$ is positive for all k , where id_k is the identity function on $\mathbb{C}^{k \times k}$.

Definition 2.8.5 (Category of completely positive maps (**CPM**))(Definition 4.3.1. from [76]). The category **CPM** is defined as follows:

- The objects are finite tuples of positive integers, $\sigma = n_1, \dots, n_k$,
- the morphisms $\sigma \rightarrow \tau$ are completely positive maps $V_\sigma \rightarrow V_\tau$ where V_{n_1, \dots, n_k} is the Hilbert space (i.e. vector space with norm) $\mathbb{C}^{n_1 \times n_1} \times \dots \times \mathbb{C}^{n_k \times n_k}$.

2.9 . Approaches to Quantum Lambda-Calculi

2.9.1 . Quantum Lambda-Calculi

The quantum lambda-calculus [62, 74, 65, 75] aims at providing a high-order lambda-calculus that is capable of representing the classical control flow introduced by measurements and, at the same time, which distinguishes the classical and the quantum data, where the quantum data is treated linearly because of no-cloning theorem. In other words, in addition to the lambda-calculus with linear types from the previous section, it necessitates bringing classical data types into the type system. It can be done by introducing the exponential constructor $!$ of the linear logic, which comes with dereliction and promotion inference rules. An example of quantum lambda calculus can be found in Definition 2.9.1. In this definition, the type $!(A \multimap B)$ can be thought of as a subtype of $A \multimap B$ which consists of duplicable terms of type $A \multimap B$, meaning that any term any term of type $!(A \multimap B)$ is also a term of type

$A \multimap B$. Although it only allows the exponential type of function types, each type A can be simulated by the function type $1 \multimap A$ which, in turn, gives us the exponential type $!(1 \multimap A)$. From the definition, type `bit` can be defined as the coproduct $1 \oplus 1$ while the terms of type `bit` are defined as `tt` = $\text{in}_r \text{skip}$ and `ff` = $\text{in}_l \text{skip}$.

Definition 2.9.1 (Quantum lambda-calculus (from [46])).

$$\begin{aligned}
(\text{Type}) \ A, B & ::= \mathbf{qubit} \mid A \multimap B \mid !(A \multimap B) \mid 1 \mid A \otimes B \mid A \oplus B \mid A^l \\
(\text{Term}) \ M, N, P & ::= x \mid \lambda x^A. M \mid MN \mid \text{skip} \mid M; N \mid M \otimes N \\
& \quad \mid \text{let } x^A \otimes y^B = M \text{ in } N \mid \text{in}_l M \mid \text{in}_r M \\
& \quad \mid \text{match } P \text{ with } (x^A : M \mid y^B : N) \mid \text{split}^A \\
& \quad \mid \text{letrec } f^{A \multimap B} x = M \text{ in } N \mid \text{meas} \mid \text{new} \mid U
\end{aligned}$$

The definition contains several new types (the type constant 1 , the co-product type $A \oplus B$, and the finite list A^l of type A) and new terms (`skip` for the type 1 and the concatenation of terms $N; M$, $\text{in}_l M$ and $\text{in}_r M$ for the introduction inference rules for \oplus , `match P with $(x^A : M \mid y^B : N)$` for the elimination rule of \oplus , `split A` for the list A^l , and `letrec $f^{A \multimap B} x = M$ in N` for recursion). Equipped with these new features, the language becomes more expressive in that more computations can be represented in the language. In particular, with the help of the finite list and the recursion, one can now represent programs that may not terminate, as shown in Example 2.9.1. Its intended meaning is that, given a qubit q , toss a coin until the head comes out while appending one qubit entangled to the original qubit q at each toss.

Example 2.9.1 (Cointoss example from [46]).

$$\begin{aligned}
& \text{letrec } f^{\mathbf{qubit} \multimap \mathbf{qubit}^l} q = \\
& \quad \text{if cointoss then } (q :: \text{nil}) \\
& \quad \text{else } (\text{let } x^{\mathbf{qubit}} \otimes y^{\mathbf{qubit}} = (\text{entangle } q) \text{ in } x :: fq) \\
& \text{in } (\lambda q^{\mathbf{qubit}}. fq)
\end{aligned}$$

where `nil` = $\text{in}_l \text{skip}$; `cointoss` = $\text{meas}(H(\text{new}(\text{tt})))$; `entangle` = $\lambda x^{\mathbf{qubit}}. N_c(x \otimes (\text{new}(\text{ff})))$; and `if P then M else N` and `$M :: N$` are abbreviations for `match P with $(x^1 : N \mid y^1 : M)$` and $\text{in}_r (M \otimes N)$, respectively.

Similar to the linear quantum lambda-calculus, the operational meaning of the term can be understood based on the QRAM model of computation as computation with side-effects that update the quantum state through the quantum co-process while the control flow of execution depends on the probabilistic outcome of the measurements. Hence, the operational semantics can be given to quantum lambda calculus as an abstract machine over the triple,

called quantum closure, $[Q, L, M]$, where Q represents the quantum state, L is the bijection between the free variables in M and the index of the qubits in the quantum state, and M is a term. Reduction rules over the quantum closure raise an equation theory of the quantum closure, although the sequence of reductions now has the possibility of having an infinite length. Moreover, the reduction rules are probabilistic. Multiple reduction rules can be applied to the same quantum closure where one reduction rule is taken according to a probability distribution. In Example 2.9.1, each time a coin is tossed, there are two possibilities of the outcome, and the term may reduce to any finite-length list of qubits.

The denotational semantics of quantum lambda-calculus relies on linear logic's categorical semantics. The function type \multimap , the exponential type $!$, the type constant 1 , and the tensor product $(- \otimes -)$ are derived from the monoidal closed category with the comonoid structure of the category.

However, to model new features of the language, we need extra structures in the category. Firstly, the category needs a bifunctor $(- \oplus -)$ for the co-product type \oplus of the language. Moreover, the list type A^l of type A implies that the category is required to interpret infinite sum. Secondly, regarding the recursion, we need to find a morphism representing the possibly infinite reduction sequence (i.e., an infinite sequence of morphisms). One way of doing it is to let the set of morphisms between any two objects be a complete partial order where the supremum of the sequence of morphisms (of the reduction of recursion) is picked for the denotation of the recursion.

In [46], the authors (Michele Pagani, Peter Selinger, and Benoît Valiron) provide a concrete categorical semantics of the quantum lambda-calculus in Definition 2.9.1 by extending the **CPM** category in Definition 2.8.5. In order to model the exponential, which is not sensitive to the order, the categorical model is based on the permutation groups and the completely positive maps, which are invariable under the permutation (Definition 2.9.3). Then, each hom-set of **CPMs** is extended by D-completion (Definition 2.9.5) so that each indexed family of morphisms has a sum defined by least upper bound (category $\overline{\mathbf{CPMs}}$ in Definition 2.9.6). Finally, the category $\overline{\mathbf{CPMs}}^\oplus$ is the completion of the category $\overline{\mathbf{CPMs}}$ in the manner of the completion of CPM category, so that the objects represent mixed states (Definition 2.9.7).

In specific, the letrec constructor is interpreted by using the fixed point operator Y as follows:

- let ϕ be a morphism in the set $\overline{\mathbf{CPMs}}^\oplus(!C \otimes !A, !A)$,
- define the morphism $\phi^n \in \overline{\mathbf{CPMs}}^\oplus$ as:

$$\begin{aligned}\phi^0 &:= !C \xrightarrow{w;!0} !A \\ \phi^{n+1} &:= !C \xrightarrow{c} !C \otimes !C \xrightarrow{\text{id} \otimes \phi^n} !C \otimes !A \xrightarrow{\phi} !A\end{aligned}$$

where $w : !C \rightarrow 1$, $!0 : 1 \rightarrow !A$, and $c : !C \rightarrow !C \otimes !C$ come from the comonoid structure.

- $Y(\phi)$ is defined as the least upper bound of the set $\{\phi^n\}$ which is directed complete.

Definition 2.9.2 (Permutation groups from [46]). Let S_n be the symmetric group of degree n , i.e., the group of permutations of $n = \{0, \dots, n-1\}$. Any permutation $g \in S_n$ gives rise to a matrix $P_g \in \mathcal{C}^{n \times n}$, defined by $P_g(e_i) = e_{g(i)}$, where e_i is the i th standard basis vector.

An action of g on $\mathcal{C}^{n \times n}$ is defined by $g \circ M := P_g M P_g^{-1}$. Moreover, for a subgroup $G \subseteq S_n$, we define

$$G \circ M := \frac{1}{\#G} \sum_{g \in G} g \circ M$$

where $\#G$ is the number of elements of G .

Definition 2.9.3 (Category **CPMs** from [46]). The category **CPMs** is the category defined with the following data:

- objects are the subgroups $G \subseteq S_n$ of the groups of permutations S_n , and
- a morphism from $G \subseteq S_n$ to $H \subseteq S_m$ is a completely positive map $f : \mathcal{C}^{n \times n} \rightarrow \mathcal{C}^{m \times m}$ which is invariant under the actions of the two subgroups G and H , i.e., $H \circ f \circ G = f$.

Definition 2.9.4 (Scott-closedness from [46]). Given any partially ordered set (poset) P , a subset $S \subseteq P$ is *Scott-closed* if it is down-closed and for every directed $I \subseteq S$, if the least upper bound $\vee I$ exists in P , then $\vee I \in S$.

Definition 2.9.5 (D-completion from [46]). Let $\Gamma(P)$ be the set of Scott-closed subsets of P , which forms a dcpo (directed-complete partial order) under the subset ordering, i.e., any subset which is non-empty and such that every pair of elements has an upper bound in the subset (directed subsets) has a supremum. The *D-completion* $c(P)$ is defined to be the smallest sub-dcpo of $\Gamma(P)$ containing all sets of the form $\downarrow x = \{y : y \leq x\}$.

Definition 2.9.6 (Category $\overline{\mathbf{CPMs}}$ from [46]). The category $\overline{\mathbf{CPMs}}$ is the category which has the same objects as **CPMs** and whose homset is obtained by taking the D-completion of the homset of **CPMs**, i.e. $\overline{\mathbf{CPMs}}(G, H) := c(\mathbf{CPMs}(G, H))$ for any objects G and H . Note that the partial order of $\overline{\mathbf{CPMs}}$ is given by the Löwner order $A \sqsubseteq B$ from Definition 2.8.3.

Definition 2.9.7 (Category $\overline{\mathbf{CPMs}}^\oplus$ from [46]). The category $\overline{\mathbf{CPMs}}^\oplus$ is defined with the following data:

- objects are given by indexed families $\mathbf{A} = \{(d_a^{\mathbf{A}}, G_a^{\mathbf{A}})\}_{a \in |\mathbf{A}|}$ where the index set $|\mathbf{A}|$ is called the web of \mathbf{A} and, for every $a \in |\mathbf{A}|$, $d_a^{\mathbf{A}}$ is a positive integer, and $G_a^{\mathbf{A}}$ a subgroup of permutations of degree $d_a^{\mathbf{A}}$, called respectively the dimension and the permutation group of \mathbf{A}_a ; and
- morphisms from \mathbf{A} to \mathbf{B} are matrices ϕ indexed by $|\mathbf{A}| \times |\mathbf{B}|$ and such that $\phi_{a,b} \in \overline{\mathbf{CPMS}}(G_a^{\mathbf{A}}, G_b^{\mathbf{B}})$.

To summarize, in terms of syntax, a quantum lambda-calculus is a lambda-calculus (e.g., Definition 2.4.1) extended with constant terms for quantum operators—initialization, unitary maps, and measurement—and exponential constructors. Similarly, the type system of quantum lambda-calculus is an extension of linear logic with quantum types. It capitalizes on linear logic to account for the non-duplicability of quantum data. In other words, a well-typed terms under a given context can be physically realized without violating the no-cloning theorem. However, quantum lambda-calculi are limited in their expressivity: they do not allow the manipulation of quantum circuits as first-class objects, limiting their use for representing non-trivial quantum algorithms.

2.9.2 . Quantum Circuit Description Language

A later trend of quantum programming languages aims at answering this problem, offering the possibility to manipulate and reason on quantum circuits as first-class objects. In general, a circuit constant has a type of $\text{Circ}(A, B)$ where A and B refer to the pattern type of the input and output of the circuit. A circuit constant is internally defined as a composition of unitary maps and initializations and measurements of qubits, while its internal structure is hidden from the programmer. In other words, the circuit is encapsulated inside some circuit constant or boxed object constructed from classical computation with circuit constants. In specific, a circuit constant has type $\text{Circ}(A, B)$ with appropriate pattern types A and B , and the box operator has type $(A \multimap B) \multimap \text{Circ}(A, B)$ which is a function from the function type $(A \multimap B)$ to the type of circuit constant $\text{Circ}(A, B)$.

A programming language may also provide circuit-level operators over the circuit objects, which include unbox, run, gate-count, control, and reverse operators. These operators help programmers code practical quantum algorithms by hiding the concrete interpretation of the operators. Moreover, useful operators can be added to the language by need later. Among the circuit-level operators, the unbox operator is utilized to send the circuit object to the quantum co-processor by linking the input of the circuit to the corresponding variables in the given pattern. For this reason, the unbox operator is sometimes called apply. Contrary to the box, the unbox operator has type $(\text{Circ}(A, B) \otimes A) \multimap B \sim \text{Circ}(A, B) \multimap (A \multimap B)$ by Currying.

Concretely, the internal definition of a circuit constant consists of a quantum circuit with linking functions of the input or output pattern required for the unbox operator. A quantum circuit can be represented as a diagram of boxes and wires, the sequential composition of quantum operators, or a completely positive map, depending on the context. Then, the pattern represents the collection of wires which reflects the structure of the composition so that one can bind two patterns constructively. The Pattern type is then the composition of the type of each wire in the pattern with the same structure. Next, in this section, we present two examples of quantum circuit description languages—QWIRE and Quipper.

QWIRE [47, 50, 51] is an embedded language that describes the quantum circuit and interacts with the host language into which it is embedded. The language consists of the circuit description part and the host language part. The circuit description part answers the question of how to represent a quantum circuit while the host language is extended from an existing language. In the language, a quantum circuit is created by composing gates and measurements (lift), unboxing a boxed circuit, or concatenating two quantum circuits. It also explains how to normalize quantum circuits. In QWIRE, the lift constructor from the circuit description part implies the dynamic lifting, allowing the host language to use the values as a resource obtained by measuring a quantum state.

The host language is extended with two operators box and run. First, the box operator transforms a circuit into the boxed circuit, which is a first-class object of type $\text{Circ}(W_1, W_2)$ where W_1 and W_2 represent the input and output wire types, respectively. Next, the run operator refers to the simulation of a circuit, which brings probabilistic side-effects of computation. The language does not specify the implementation of the run operator, which means that it can be either interaction with quantum co-processor or classical simulation. For the classical simulation, the operational meaning of the quantum circuit is formally defined by denotational semantics. Accordingly, the operational semantics of the host language is extended with the probabilistic reduction step of the run operator.

In an implementation QWIRE over the proof assistant Coq, the host language is equipped with all features provided by the internal logic of COQ (Calculus of Inductive Construction). In particular, the reduction of the term is given by the simplification provided by Coq. Moreover, each term has a type given by the internal logic, which is an extension of a dependent type system. An example of what we can do with the dependent type system of Coq is to define a dependent circuit or parameterized circuit. It is essential to obtain a scalable quantum programming language since quantum algorithms are defined over an arbitrary size of the input (number of qubits). Furthermore, the functional meaning of the program in the host language of QWIRE

can be specified by using the dependent type system (or internal logic) of Coq. In this way, any quantum algorithm written in QWIRE can be specified by a type judgement and verified by a type derivation.

Quipper [28, 27, 68] is another quantum circuit description language that is scalable and expressive and which is used to implement various realistic quantum algorithms ranging from Binary Welded Tree (BWT) to Quantum Linear Systems (QLS). As introduced before, Quipper is an embedded language whose implementation is embedded in Haskell. Quipper consists of a language for circuit description, a procedural description of an extension of the quantum circuit with ancillas. It asserts circuit operators, run and box operators, and quantum data types. The type of circuits (or boxed subcircuits) is implemented as a monad in Haskell, which comes with unit and multiplication functions. Then, we can define the box operator as a function from the function type to the circuit type or the computation type because the computation of the program outputs a circuit.

An important programming paradigm of Quipper is to have two phases of computation: circuit generation and circuit execution. In the circuit generation phase, the program is transformed into a quantum circuit while consuming parameters which include input-size, error-thresholds, etc. Circuit execution may occur in an actual quantum processor or a classical computer by simulating the circuit generated from the circuit generation phase. In the circuit execution phase, the simulation transforms the input state into the output state.

Consequently, in Quipper, there are two different notions of resource-parameter and state. Although there is no difference between the parameter and the state appearing in the program, their types may differ. As an example, the Boolean parameter has type `Bool` while the Boolean state has type `Bit` and the qubit state has type `Qubit`. In this context, dynamic lifting is a function from `Bit` to `Bool`. Based on the two-phase paradigm of Quipper, dynamic lifting is an expensive operation since the circuit generation phase is suspended until the sub-circuit is executed.

Moreover, parameter and state have different properties; the parameter is classical data, while the state can be quantum (or wire in the circuit diagram). In other words, the parameter is a non-linear resource while the state is a linear resource. Unfortunately, Haskell does not provide linear types in nature, implying that the Haskell type checker cannot distinguish physically non-realizable programs. This limit has led to the study of ProtoQuipper, a formal language designed for quantum circuit description.

ProtoQuipper [56] is a formalization of a part of Quipper based on an extension of primitive quantum lambda-calculus without general coproduct types and recursion by adding circuit type ($\text{Circ}(T, U)$) and circuit operators (rev , unbox , and box^T) as shown in Definition 2.9.8. As quantum lambda-

calculi, it capitalizes on linear logic to forbid the duplication of quantum data, and both quantum bits and quantum circuits are considered first-order objects.

The first ProtoQuipper from Neil Ross's thesis comes with a sound operational semantics over the closure, which is the pairs (C, a) where C refers to a circuit and a refers to a term. The definition of closure is different from that of quantum lambda-calculus since the operational semantics of ProtoQuipper focuses on the circuit generation phase of the computation of the Quipper program.

Definition 2.9.8 (Definition 8.2.1. and 8.2.2. from [56]). Types and terms of ProtoQuipper are defined as follows:

(types) $A, B ::= \mathbf{qubit} \mid 1 \mid \mathbf{bool} \mid A \otimes B \mid A \multimap B \mid !A \mid \mathbf{Circ}(T, U)$
 (terms) $a, b, c ::= x \mid q \mid (t, C, a) \mid \mathbf{True} \mid \mathbf{False} \mid \langle a, b \rangle \mid * \mid ab \mid \lambda x.a$
 $\mid \mathit{rev} \mid \mathit{unbox} \mid \mathit{box}^T \mid \text{if } a \text{ then } b \text{ else } c \mid \text{let } * = a \text{ in } b$
 $\mid \text{let } \langle x, y \rangle = a \text{ in } b$

where (t, C, a) refers to the circuit constant; and rev , unbox , and box^T refer to circuit operators.

Later, several extensions of ProtoQuipper have been proposed, which formalize different features of programming languages like recursion and dependent type systems. Concerning this thesis, we study the dynamic lifting in quantum circuit description language and formalize it in a language extended from ProtoQuipper. Moreover, the categorical semantics of different versions of Quipper have been studied. We also study the categorical semantics of the language, which we define to formalize dynamic lifting.

2.10 . Categorical models of quantum computation

2.10.1 . Completely positive maps

Completely positive maps, which we have introduced in Section 2.8 have been utilized as the semantics of quantum circuit description languages as well. To summarize the semantics briefly, a square matrix $A \in \mathbb{C}^{n \times n}$ is called hermitian if $A = A^*$. A positive, or positive semidefinite, matrix is a hermitian matrix such that $u^* A u \geq 0$ for all $u \in \mathbb{C}^n$. A density matrix is a positive matrix formed in a particular way. It represents the state of a system following a sequence of quantum operations (unitary maps and measurements).

More specifically, we have introduced the quantum state in Section 2.1.1 as a normalized vector q called pure state in the vector space of the quantum state. Then, the density matrix of the quantum state q is defined as $q q^*$. In addition, measurement creates a probability distribution of states which is called mixed state. A mixed state is given as a family of pairs of a pure state

q_i and its probability p_i where $1 \leq i \leq k$, and then the density matrix for the given mixed state is defined as $p_1 q_1 q_1^* + \dots + p_k q_k q_k^*$.

Operations over the density matrix are defined for both unitary maps and measurements. A unitary map U maps a density matrix (pure and mixed states) A to a density matrix UAU^* . For the measurement, the resulting density matrix is the sum of each density matrix corresponding to the measurement state. In specific, for a given density matrix $\sum_{1 \leq i \leq k} p_i q_i q_i^*$, let us assume that the measurement applied to the pure state q_i results in the state q_i^1 with probability p_i^1 and in the state q_i^2 with probability p_i^2 . Then, the resulting density matrix by the measurement is the sum of all resulting states of measurement at the pure state, which is $\sum_{1 \leq i \leq k} p_i (p_i^1 q_i^1 (q_i^1)^* + p_i^2 q_i^2 (q_i^2)^*)$.

These quantum operators are interpreted as superoperators (Definition 2.10.1), which are linear maps from a list of square matrices to a list of square matrices that preserve the positive elements even if the maps are extended with identity maps by tensor product (i.e., completely positivity) and satisfies the trace condition. Moreover, the superoperator preserves Löwner partial order (Definition 2.8.3) and the upper bounds of increasing sequences. This forms the category of quantum computation, and its CPO-enrichment can be used to model recursion [59].

Definition 2.10.1 (Completely positive operator, superoperator (from [59])).

Let $F : V_\sigma \rightarrow V_{\sigma'}$, where $\sigma = n_1, \dots, n_s$ and $\sigma' = m_1, \dots, m_t$ are lists of natural numbers (i.e., *signatures*), be a linear function from the vector space $V_\sigma = \mathbb{C}^{n_1 \times n_1} \times \dots \times \mathbb{C}^{n_s \times n_s}$ to the vector space $V_{\sigma'} = \mathbb{C}^{m_1 \times m_1} \times \dots \times \mathbb{C}^{m_t \times m_t}$. Note that elements of V_σ are tuples of matrices $A = (A_1 \in \mathbb{C}^{n_1 \times n_1}, \dots, A_s \in \mathbb{C}^{n_s \times n_s})$. Then, we say that F is *positive* if $F(A)$ is positive for all positive $A \in V_\sigma$ meaning that each element A_i of A is positive. We say that F is *completely positive* if $\text{id}_\tau \otimes F : V_{\tau \otimes \sigma} \rightarrow V_{\tau \otimes \sigma'}$ is positive for all signatures $\tau = o_1, \dots, o_u$, where $\tau \otimes \sigma = o_1 n_1, \dots, o_1 n_s, \dots, o_u n_1, \dots, o_u n_s$ and $\text{id}_\tau \otimes F$ is defined on basis elements by $(\text{id} \otimes F)(A \otimes B) = \text{id}(A) \otimes F(B)$. Finally, F is called a *superoperator* if it is completely positive and satisfies the following trace condition: $\text{tr}(F(A)) \leq \text{tr}(A)$, for all positive $A \in V_\sigma$, where $\text{tr}(A) = \sum_i \text{tr}(A_i)$.

The fact that a quantum state can be represented as a positive matrix called a density matrix explains why we are interested in positive matrices and the positive linear maps, which preserve positive matrices. Completely positive maps form the canonical semantics of quantum computation [45]. Selinger has shown in 2004 [57] that one can formalize it as a semantics for first-order languages, and later exploration showed that completely positive maps could be used to build semantics of quantum lambda-calculus [58, 64].

One of the limitations of completely positive maps and positive matrices is the finite-dimensionality of the model: this makes it impossible to represent duplicable data or inductive datatypes. Several extensions have been devised,

either based on suitable algebraic extensions such as C^* or von Neumann algebras [79, 78], or using categorical constructs [46, 40, 41].

2.10.2 . Semantics of quantum channels

The meaning of quantum channels may vary across different research domains like quantum information theory and quantum operator theory. This thesis follows the definition of quantum operator theory and defines the quantum channel as an algebraic structure. An operator algebra consists of a signature for the algebra and an equation theory over the operators. A quantum channel is then defined by an equivalence class of operators.

In order to define a practical operator algebra of quantum channels, one needs to know what it represents, namely, in quantum computation. Based on the formalism of Dirac and Von Neumann, quantum mechanics can be explained in C^* -algebras. In this formalism, observables of a quantum system form self-adjoint operators on a Hilbert space, i.e., a Banach space equipped with an inner product that is consistent with the norm. The observables of n -dimensional Hilbert space for any natural number n form a C^* -algebra. One can then define a state as a function from the self-adjoint elements of the C^* -algebra to reals in $[0, 1]$.

In this context, quantum computation corresponds to the maps between C^* -algebras that preserve positive elements (positive), hence, the self-adjoint elements since self-adjoint elements are positive. Moreover, the map needs to preserve the unit element (unital) and needs to be able to be extended with other disconnected parts of the system (completely positive). This forms a category of C^* -algebras with unital completely positive maps, $Cstar_{CPU}$, whose objects are the C^* -algebras over different Hilbert spaces, and morphisms are the unital completely positive maps between these algebras.

For quantum computation with measurement, one needs to be able to represent observables that depend on the measurement results. In category theory, coproduct, or direct sum, represents this non-determined set of observables. Coproduct in C^* -algebras is defined as a juxtaposition of the observables corresponding to different measurement results. Now, quantum computation with measurement, assuming that there has been no measurement made before, becomes completely positive and unital maps from a C^* -algebra to a coproduct of C^* -algebras. In general, quantum computation beginning with measurements made before can be represented as the juxtaposition of such maps for each C^* -algebra from the initial stage.

A quantum channel in an operator algebra is usually represented by an operator parameterized by the input variables. Intuitively, these operators represent the observables in the C^* -algebra of the corresponding quantum system. As an example of an operator algebra, in [70], Sam Staton defines the algebraic structure of quantum computation as in Definition 2.10.2.

Definition 2.10.2 (Algebraic structure of quantum computation (Definition 3 from [70])). There are three accessor functions **new** for the initialization of qubit, **meas** for the measurement, and **apply_U** for the application of unitary map U defined as follows:

$$\mathbf{new} : (0 \mid 1), \mathbf{meas} : (1 \mid 0, 0) \text{ and } \mathbf{apply}_U : (n \mid n)$$

where $o : (\tau \mid \sigma)$ means that the operator o takes input $\tau = n_1, \dots, n_s$ and output $\sigma = m_1, \dots, m_t$ that represents the list of quantum states of n_i -qubits and m_j -qubits, respectively, and U is a unitary operator over 2^n dimension state space.

Each quantum operator is defined by composition of the accessor functions in the following manner:

$$t ::= \begin{cases} \mathbf{new}(a.t), \\ \mathbf{meas}(a, t, u), \\ \mathbf{apply}_U(a_1 \dots a_n, b_1 \dots b_n.t) \end{cases}$$

where **new**($a.t$) refers to the creation of a new qubit a which passes the qubit to the operator t ; **meas**(a, t, u) refers to the measurement of the qubit a which continues to either t or u depending on the measurement outcome; and **apply_U**($a_1 \dots a_n, b_1 \dots b_n.t$) refers to the application of unitary map U to the qubits $a_1 \dots a_n$ and passes these qubits to the operator t after binding the qubits to the names $b_1 \dots b_n$.

There are three types of operator, each of which is parameterized by specific numbers of qubits: firstly, new is an operator that creates a qubit and passes it through the rest of the operator while not taking any input; secondly, meas takes the name of one qubit as input, which is measured and disappears, and passes through different parts of the rest of the operator depending on the measurement result; and, lastly, apply_U applies a unitary operator U to the n qubits designated by names in the input and passes the new names of these qubits through the rest of the operator.

Furthermore, Sam Staton defines unitary operator as shown in Definition 2.10.3.

Definition 2.10.3 (Construction of the unitary operator from [70]).

$$U ::= \dots \mid U_1 \otimes U_2 : (n_1 + n_2 \mid n_1 + n_2) \mid D(U_1, \dots, U_k) : (n \mid n)$$

where \dots means a choice of an unitary operator set; n_i refers to the size of U_i ; $U_1 \otimes U_2$ means the tensor product of the two unitary operators U_1 and U_2 ; and $D(U_1, \dots, U_k)$ refers to the block diagonal matrix created by the unitary operators U_i of size n_i such that n is the size of the new operator such that $2^n = \sum_{i \leq k} 2^{n_i}$. Note that the last constructor of diagonal matrix is defined only for unitary operators such that there exists n that satisfies the given condition on the size.

To summarize, the unitary operator set is the closure of the chosen unitary operator set over the tensor product; the group product of the unitary operators of the same dimension; and the block diagonal composition needs to be complete concerning finite-dimensional unitary operators. For instance, the following unitary operators form a complete set of unitary operators over the compositions explained above.

$$\left\{ \text{id}_0 = (1), X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Z^\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}, H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right\}$$

where $\text{id}_0 : (0 \mid 0)$ and $X, Z^\theta, H : (1 \mid 1)$

The algebraic operator theory comes with axioms that equate the operators based on syntactic pattern matching. In the paper [70], Sam Staton proposes an algebraic theory which consists of the axioms in Definition 2.10.4.

Definition 2.10.4 (Algebraic theory of quantum computation (Sam Staton)). An algebraic theory of quantum computation is defined by the following equations over the operators from Definition 2.10.2.

1. $\mathbf{apply}_X(a, a.\mathbf{measure}(a, x, y)) = \mathbf{measure}(a, y, x)$
where X is from the unitary operator set
2. $\mathbf{measure}(a, \mathbf{apply}_U(\vec{b}, \vec{b}.x(\vec{b})), \mathbf{apply}_V(\vec{b}, \vec{b}.y(\vec{b})))$
 $= \mathbf{apply}_{D(U,V)}(a :: \vec{b}, (a :: \vec{b}).\mathbf{measure}(a, x(\vec{b}), y(\vec{b})))$
3. $\mathbf{apply}_U(\vec{a}, \vec{a}.\mathbf{discard}_n(\vec{a}, x)) = \mathbf{discard}_n(\vec{a}, x)$ where $\mathbf{discard}_n(\vec{a}, x)$ refers to the operator which discards the qubits \vec{a} and continues the operator x which is defined inductively as follows:

$$\mathbf{discard}_0(-, t) = t$$

$$\mathbf{discard}_{n+1}((a :: \vec{b}), t) = \mathbf{measure}(a, \mathbf{discard}_n(\vec{b}, t), \mathbf{discard}_n(\vec{b}, t))$$

4. $\mathbf{new}(a.\mathbf{measure}(a, x, y)) = x$
5. $\mathbf{new}(a.\mathbf{apply}_{D(U,V)}((a :: \vec{b}), (a :: \vec{b}).x(a :: \vec{b})))$
 $= \mathbf{apply}_U(\vec{b}, \vec{b}.\mathbf{new}(a.x(a :: \vec{b})))$
6. $\mathbf{apply}_{\text{swap}}((a :: b), (a :: b).x(a :: b)) = x(b :: a)$ where

$$\text{swap} = D(\text{id}_0, X, \text{id}_0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7. $\mathbf{apply}_I(\vec{a}, \vec{a}.x(\vec{a})) = x(\vec{a})$

8. $\mathbf{apply}_{UV} = \mathbf{apply}_U(\vec{a}, \vec{a}.\mathbf{apply}_V(\vec{a}, \vec{a}.x(\vec{a})))$
9. $\mathbf{apply}_{U \otimes V}((\vec{a} + + \vec{b}), (\vec{a} + + \vec{b}).x(\vec{a} + + \vec{b}))$ where $\vec{a} + + \vec{b}$ refers to the list of qubits obtained by concatenation of the lists of qubits \vec{a} and \vec{b} .
 $= \mathbf{apply}_U(\vec{a}, \vec{a}.\mathbf{apply}_V(\vec{b}, \vec{b}.x(\vec{a} + + \vec{b})))$

Indeed, the operators of the theory need more information about the variables. Each term of the theory is constructed by the operator (Definition 2.10.2) with a sequence of quantum variables Δ of type qubit and a sequence of second-order variables Γ of functions from qubits to qubits. Following the notation of Sam Staton [70], each term is represented by $\Gamma \mid \Delta \vdash t$ where t is an operator from Definition 2.10.2. For the first-order theory, Γ will be empty. Since Δ is represented as a sequence, each permutation on the sequence of variables Δ creates a different term, although their intended meaning should be the same. In models of algebraic theories (like the category of sets and functions \mathbf{Set} and the category of C^* -algebras), the effect of permutation is normally represented by the cartesian product structure of the category of bijection and the category of operators.

In his paper [70], Sam Staton shows that the algebraic theory of quantum computation proposed in the paper is fully complete for the model of quantum computation (the category \mathbf{Cstar}_{CPU} introduced earlier). In particular, each operator can be interpreted as a linear map (i.e., matrix) between C^* -algebras based on the interpretation rules in Definition 2.10.5. Each operator term $\emptyset \mid \Delta \vdash t$ is interpreted by a matrix of size $2^{|\Delta|} \times 2^{|\Delta|}$. Note that each interpretation is paired with an action of permutation over the variables Δ . The action of permutation is interpreted by the multiple applications of the swap operator (where each swap operator changes the order of two adjacent variables), following the interpretation of the unitary operator in Definition 2.10.5.

Definition 2.10.5 (Interpretation of operators). The following rules interpret each operator.

$$\begin{aligned} \llbracket \mathbf{new}(a.x(a)) \rrbracket &= a_{1,1} \text{ where } \llbracket x(a) \rrbracket = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \\ \llbracket \mathbf{meas}(a, x, y) \rrbracket &= \begin{pmatrix} \llbracket x \rrbracket & 0 \\ 0 & \llbracket y \rrbracket \end{pmatrix} \\ \llbracket \mathbf{apply}_U(\vec{a}, \vec{b}.x(\vec{b})) \rrbracket &= \llbracket U \rrbracket^* \llbracket x(\vec{b}) \rrbracket \llbracket U \rrbracket \end{aligned}$$

Next, the unitary operator is interpreted based on the following rules together with the conventional matrix definition of the constant unitary opera-

tors like id_0 , X , Z^θ , and H .

$$\begin{aligned} \langle U_1 \otimes U_2 \rangle &= \begin{pmatrix} \langle U_1 \rangle_{1,1} \langle U_2 \rangle & \dots & \langle U_1 \rangle_{1,n} \langle U_2 \rangle \\ \vdots & \ddots & \vdots \\ \langle U_1 \rangle_{n,1} \langle U_2 \rangle & \dots & \langle U_1 \rangle_{n,n} \langle U_2 \rangle \end{pmatrix} \\ \langle D(U_1, \dots, U_k) \rangle &= \begin{pmatrix} \langle U_1 \rangle & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \langle U_k \rangle \end{pmatrix} \end{aligned}$$

where the $\langle U_1 \rangle$ is a matrix of size $n \times n$ in the tensor product case.

Given this, the full completeness of the algebraic theory of quantum computation means that: first, any morphism in the model of quantum computation, which is a unital completely positive map, can be represented as an operator, where the interpretation of the operator is the original linear map in the category; and, secondly, the equality of two operators is derivable within the algebraic theory in Definition 2.10.4 whenever the two operators have the same interpretation as unital completely positive maps.

The operator algebra provides a good language of quantum channels which is fully complete. However, the language only allows variables up to first-order functions (i.e., operator) in the program. Hence, it does not support, for example, general recursion, which relies on the high-order abstraction. In order to obtain a high-order language of quantum channels, we instead begin from ProtoQuipper, which is based on quantum lambda-calculus, and adjust the categorical model of ProtoQuipper to get a model of quantum channels instead of quantum circuits.

2.10.3 . Models of circuit-description languages

ProtoQuipper-M by Rios and Selinger [55]

In this paragraph, we summarize the categorical model of the quantum circuit description language ProtoQuipper-M by Francisco Rios and Peter Selinger [55]. A circuit description language allows programmers to construct circuits using circuit-level operators. At the same time any circuit can be simulated according to the intended meaning of each operator that constructs the circuit. Consequently, a program needs to deal with two stages of computation. A program generates a circuit from the first level of computation, and the circuit generates a state in the second computation. In a circuit description language, the computation is one-directional, meaning that the computation of the circuit does not affect the circuit generation.

An essential feature of a scalable circuit description language is the capability to represent not only a single circuit but also a family of circuits, or a parameterized circuit. This feature enables programmers to express more

complex structures in the language while introducing complexity to a program's semantics. For instance, a family of circuits can be parameterized by another family of circuits. The semantics of the language should be able to interpret compositions or executions of such families of circuits.

In [55], the authors show that the functor category $\mathbf{C}^{2^{\text{op}}}$ (whose objects are functors and morphisms are natural transformations) from $\mathbf{2}^{\text{op}}$ (which is the category with two objects 0 and 1, and a single non-identity morphism $1 \rightarrow 0$) to some category \mathbf{C} can interpret parameters and states, as well as parameterized circuits in the object. Here, parameters are the values known at circuit generation time, and states are the values known at circuit execution time. Therefore, states cannot influence parameters, but parameters can influence states. This structure encoded in the functor category $\mathbf{C}^{2^{\text{op}}}$ comes from the unique non-identity morphism of the category $\mathbf{2}^{\text{op}}$.

Specifically, a cartesian model [55] of parameters and states $\mathbf{Set}^{2^{\text{op}}}$ is the functor category from the category $\mathbf{2}^{\text{op}}$ to the category of sets and functions \mathbf{Set} defined with the following data:

- objects: functors from the category $\mathbf{2}^{\text{op}}$ to the category \mathbf{Set} represented as (A_0, A_1, a) where A_0 and A_1 are the sets corresponding to the two objects of $\mathbf{2}^{\text{op}}$ and a is the corresponding morphism from A_1 to A_0 ,
- morphisms: natural transformations $f : A \rightarrow B$ between the functors $A = (A_0, A_1, a)$ and $B = (B_0, B_1, b)$ from $\mathbf{2}^{\text{op}}$ to \mathbf{Set} such that $b \circ f_0 = f_1 \circ a$, i.e. $b(f_0(x)) = f_1(a(x))$ for all $x \in A_1$.

An object of the category is equivalently represented as $(A_0, (A_x)_{x \in A_0})$ where each A_x is the subset of A_1 such that $a(x') = x$ for all $x' \in A_x$ [55]. Then, the pair (x, s) is called a generalized element of the object $A = (A_0, (A_x)_{x \in A_0})$ if x is in A_0 and s is in A_x . Now, we can consider x , and s , of a generalized element of the object as the parameter, and the state, respectively.

Then, we can define particular types of objects in the category $\mathbf{Set}^{2^{\text{op}}}$ called parameter objects and state (or simple) objects. A parameter object is defined by the form (A, A, \mathbf{id}) , an example of which is $\mathbf{bool} = (\{0, 1\}, \{0, 1\}, \mathbf{id})$, or equivalently, represented as the set of generalized elements $\mathbf{bool} = \{(0, 0), (1, 1)\}$. Next, a state object is of the form $(\{*\}, A, a)$. For example, $\mathbf{bit} = (\{*\}, \{0, 1\}, (\{0 \mapsto *, 1 \mapsto *\}))$, or equivalently, $\mathbf{bit} = \{(*, 0), (*, 1)\}$.

The category $\mathbf{Set}^{2^{\text{op}}}$ is cartesian closed, which means that there is an isomorphism between the hom set $\mathbf{Hom}(X, Z^Y)$ and the hom set $\mathbf{Hom}(X \times Y, Z)$ for any objects X, Y , and Z . Therefore, we can interpret the simply-typed lambda calculus in it: where the exponential objects interpret functions, e.g. $\llbracket \text{init} : \mathbf{bool} \rightarrow \mathbf{bit} \rrbracket = \mathbf{bit}^{\mathbf{bool}}$.

The cartesian model of parameters and states $\mathbf{Set}^{2^{\text{op}}}$ allows duplication of states since there exist morphisms $A \rightarrow A \times A$ for any object A . However,

states generated by quantum circuit description languages should not be duplicated to conform with the no-cloning theorem of quantum physics. Therefore, the authors define another categorical model of parameters and states denoted by $\overline{\overline{M}}$ in the following way:

1. define a symmetric monoidal category M to interpret the quantum states and quantum circuits.
2. define a symmetric closed, product-complete, monoidal category \overline{M} where M embeds into \overline{M} . The monoidal closed structure of \overline{M} allows us to interpret high-order functions in \overline{M} .
3. define a category $\overline{\overline{M}}$ for parameterized circuits with:
 - objects: pairs $A = (X, (A_x)_{x \in X})$, where X is a set and $(A_x)_{x \in X}$ is an X -indexed family of objects of \overline{M} and the element A_x is called the fiber of A over x .
 - morphisms: $f : (X, (A_x)_{x \in X}) \rightarrow (Y, (B_y)_{y \in Y})$ is a pair $(f_0, (f_x)_{x \in X})$, where $f_0 : X \rightarrow Y$ is a function and each $f_x : A_x \rightarrow B_{f_0(x)}$ is a morphism of \overline{M} .

The category $\overline{\overline{M}}$ contains (by an inclusion) the symmetric monoidal category M , which models general circuits, an example of which is quantum circuits. This category has nice properties to interpret classical computation; for instance, it has an initial object and coproduct and is symmetric monoidal closed. Moreover, it has an inclusion functor $p : \mathbf{Set} \rightarrow \overline{\overline{M}}$ and its adjoint $\flat : \overline{\overline{M}} \rightarrow \mathbf{Set}$, which forms a categorical model of linear logic with exponential $! = p \circ \flat$ and which gives an interpretation of box and unbox operators. Specifically, the circuit type $\text{Circ}(T, U)$ for quantum type T and U is interpreted as $p(M(\llbracket T \rrbracket, \llbracket U \rrbracket))$ and $\text{box} : !(T \multimap U) \multimap \text{Circ}(T, U)$ and $\text{unbox} : \text{Circ}(T, U) \multimap !(T \multimap U)$ operators are defined using the following isomorphism which yields morphisms $\text{box} : !(\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \rightarrow p(M(\llbracket T \rrbracket, \llbracket U \rrbracket))$ and $\text{unbox} : p(M(\llbracket T \rrbracket, \llbracket U \rrbracket)) \rightarrow !(T \multimap U)$.

$$\flat(\llbracket T \rrbracket \multimap \llbracket U \rrbracket) \cong \overline{\overline{M}}(I, \llbracket T \rrbracket \multimap \llbracket U \rrbracket) \cong \overline{\overline{M}}(\llbracket T \rrbracket, \llbracket U \rrbracket) \cong M(\llbracket T \rrbracket, \llbracket U \rrbracket)$$

In summary, the categorical semantics formalizes a fragment of Quipper regarding the circuit construction and provides a formalized language called Proto-Quipper-M.

Finally, the question we ask in this thesis is if we can build a similar categorical semantics for the circuits extended with measurement and dynamic lifting, called quantum channels. This implies that the measurement of a quantum state produces parameters that can be used to construct other quantum channels. It must break the assumption of the one-directional computation

from the circuit generation to the circuit execution. However, the idea that we follow in this thesis is to think of a quantum channel as a tree of quantum circuits, where the tree structure is a classical parameter, which can be represented as the set of paths closed under a prefix. From this idea, we interpret a quantum channel as a parameterized circuit and try to find if it gives a categorical semantics.

Lindenhovius, Mislove, and Zamdzhiev's mixed linear-non-linear model [38, 39] Although the categorical model \overline{M} of Francisco Rios and Peter Selinger from the previous paragraph models several inductive types like natural numbers or lists, we cannot interpret any inductive types without the further assumption that the category has any colimit. In other words, not all functors of the categorical model have an initial algebra. In this section, we introduce the categorical semantics of mixed linear/non-linear type systems with inductive types and recursion of Bert Lindenhovius, Michael Mislove, and Vladimir Zamdzhiev from [39, 39].

Linear and non-linear fixpoint calculus (LNL-FPC) is a language that is similar to the quantum lambda-calculus without recursion. Mixed LNL-FPC is the same language where we do not have separate context for linear variables and non-linear variables but one type of context that mixes linear and non-linear variables. This language is extended with different types for variable types X , the fixpoint $\mu X.A$, and additional terms for folding and unfolding the type $\mu X.A$.

First of all, these new types are used to define inductive types like the natural numbers $\text{Nat} = \mu X.(I + X)$, where $I = !(0 \multimap 0)$ and $0 = \mu Y.Y$; and lists $\text{List}(A) = \mu X.(I + A \otimes X)$ for any type A . For example, a natural number is an infinite sum $\mu X.(I + X) = I + (I + (I + \dots))$ and, in order to obtain the equation, the fixpoint type is unfolded by the following equation of substitution:

$$\mu X.(I + X) = I + \mu X.(I + X).$$

More generally, the unfolding of type $\mu X.A$ is given by $A[\mu X.A/X]$ and we call the type $\mu X.A$ the folding of type $A[\mu X.A/X]$. Formally, the fold and unfold constructors correspond to typing derivations which are shown in Eq. 2.10.

$$\frac{\Theta; \Gamma \vdash m : A[\mu X.A/X] \quad \Theta, X \vdash A}{\Theta; \Gamma \vdash \text{fold}_{\mu X.A}(m) : \mu X.A} \quad \frac{\Theta; \Gamma \vdash m : \mu X.A}{\Theta; \Gamma \vdash \text{unfold}(m) : A[\mu X.A/X]} \quad (2.10)$$

where Θ and Γ refer to typing context and term context respectively; and $\Theta; X \vdash A$ is a type context relation. The type context describes the valid constructions of type $\mu X.A$.

An essential consequence of inductive types is term-level recursion. For the case of a classical type system, recursion can be defined in FPC as $\text{rec } x^A.m =$

$\alpha_m^x(\text{fold}(\alpha_m^x))$ where $\alpha_m^x = (\lambda y^{\mu X.(X \rightarrow A)}.((\lambda x^A.m)((\text{unfold}(y))y)))$. The type relation $\Theta; \Gamma \vdash \text{rec } x^A.m : A$ is derived from $\Theta; \Gamma, x : A \vdash m : A$ as follows:

$$\frac{\frac{\vdots}{\Theta; \Gamma \vdash \alpha_m^x : (\mu X.(X \rightarrow A)) \rightarrow A} \quad \frac{\vdots}{\Theta; \Gamma \vdash \alpha_m^x : (\mu X.(X \rightarrow A)) \rightarrow A}}{\Theta; \Gamma \vdash \alpha_m^x(\text{fold}(\alpha_m^x)) : A}$$

where $\Theta; \Gamma \vdash \alpha_m^x : (\mu X.(X \rightarrow A)) \rightarrow A$ is derived as follows:

$$\frac{\frac{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash y : \mu X.(X \rightarrow A)}{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash \text{unfold}(y) : (\mu X.(X \rightarrow A)) \rightarrow A} \quad \frac{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash y : (\mu X.(X \rightarrow A))}{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash (\text{unfold}(y))y : A}}{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash (\text{unfold}(y))y : A}$$

and

$$\frac{\frac{\frac{\Theta; \Gamma, x : A \vdash m : A}{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)), x : A \vdash m : A} \quad \vdots}{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash \lambda x^A.m : A \rightarrow A} \quad \frac{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash (\text{unfold}(y))y : A}{\Theta; \Gamma, y : (\mu X.(X \rightarrow A)) \vdash (\lambda x^A.m)((\text{unfold}(y))y) : A}}{\Theta; \Gamma \vdash \lambda y^{\mu X.(X \rightarrow A)}.((\lambda x^A.m)((\text{unfold}(y))y)) : (\mu X.(X \rightarrow A)) \rightarrow A}$$

Moreover, the reduction of the term $\text{rec } x^A.m$ to the term $m[(\text{rec } x^A.m)/x]$ is derived from the operational semantics as follows: given that $\text{unfold}(\text{fold}(t)) \rightarrow^* t$,

$$\frac{\frac{\frac{\alpha_m^x(\text{fold}(\alpha_m^x)) \rightarrow^* m[(\text{unfold}(\text{fold}(\alpha_m^x)))(\text{fold}(\alpha_m^x))/x]}{\alpha_m^x(\text{fold}(\alpha_m^x)) \rightarrow^* m[(\alpha_m^x)(\text{fold}(\alpha_m^x))/x]} \quad \frac{\alpha_m^x(\text{fold}(\alpha_m^x)) \rightarrow^* m[(\text{rec } x^A.m)/x]}{\text{rec } x^A.m \rightarrow^* m[(\text{rec } x^A.m)/x]}}{\text{rec } x^A.m \rightarrow^* m[(\text{rec } x^A.m)/x]}$$

where $\alpha_m^x(t) \rightarrow^* m[(\text{unfold}(t))t/x]$ is obtained as follows:

$$\frac{(\lambda y^{\mu X.(X \rightarrow A)}.((\lambda x^A.m)((\text{unfold}(y))y)))(t) \rightarrow (\lambda x^A.m)((\text{unfold}(t))t)}{\alpha_m^x(t) \rightarrow (\lambda x^A.m)((\text{unfold}(t))t)}$$

and

$$\overline{(\lambda x^A.m)((\text{unfold}(t))t) \rightarrow m[(\text{unfold}(t))t/x]}$$

Similarly, in the context of mixed linear and non-linear fixpoint calculus, recursion is defined as Definition 2.10.6 in [39]. As a result, the categorical semantics of recursion follows from the categorical model of the LNL-FPC and the inductive types.

Definition 2.10.6 (Definition of recursion (Theorem 2.3.1. and Theorem 3.0.1. from [39])). Recursion term $\text{rec } z^{!A}.m$ is defined as $(\text{unfold force } \alpha_m^z)\alpha_m^z$ where

$$\alpha_m^z = \text{lift fold } \lambda x^{! \mu X.(!X \multimap A)}.(\lambda z^{!A}.m)(\text{lift (unfold force } x)x)$$

where lift x and force x are terms for the introduction and the elimination typing derivation for exponential $!$. The typing rule

$$\frac{\Theta; \Phi, z : !A \vdash m : A}{\Theta; \Phi \vdash \text{rec } z^{!A}.m : A}$$

is derivable in LNL-FPC where Φ is non-linear and $X \notin \Theta$ and $x \notin \Phi$. Moreover, the evaluation rule

$$\frac{m[\text{lift rec } z^{!A}.m/z] \rightarrow^* v}{\text{rec } z^{!A}.m \rightarrow^* v}$$

is derivable with LNL-FPC.

For the categorical semantics of fixpoint calculus, the inductive type is denoted by using the colimit of ω -diagram to a category \mathcal{C} . Intuitively, ω -diagram represents the fixpoint of the infinite sequence of the substitution in the inductive type while the colimit internalizes the infinite sequence represented by a ω -diagram as an object of \mathcal{C} . As an example, the natural number $\text{Nat} = \mu X.(I + X)$ creates the following sequence:

$$\text{Nat} \rightarrow (I + \text{Nat}) \rightarrow (I + (I + \text{Nat})) \rightarrow \dots$$

In other words, the denotation of Nat can equally be the denotation of any term in the sequence, which can be considered infinite coproduct or, more specifically, colimit of ω -diagram when we want to include the equality of types in the sequence.

Then, the categorical semantics of LNL-FPC, called CPO-LNL model (Definition 2.10.7 from [39]), are obtained from Benton's linear category with the category CPO (a category of complete partial orders (CPO) and Scott-continuous functions) and a CPO-category L (an enriched category whose homsets have the CPO structure and the composition of morphisms is a Scott continuous function that preserves all suprema of increasing chains of morphisms).

Definition 2.10.7 (A CPO-LNL model (Definition 5.3.1. from [39])). A CPO-LNL model is given by the following data:

- A CPO-symmetric monoidal closed category $(L, \otimes, \multimap, I)$ such that:
 - L has an e-initial object 0 and all ω -colimits over embeddings;
 - L has finite CPO-coproducts, where $(- + -) : L \times L \rightarrow L$ is the coproduct functor.
- A CPO-symmetric monoidal adjunction:

$$\begin{array}{ccc} & F & \\ & \curvearrowright & \\ \text{CPO} & \perp & L \\ & \curvearrowleft & \\ & G & \end{array}$$

ProtoQuipper-D by Fu, Kishida, and Selinger [24, 23]

The dependent type system is another feature of programming language studied in the paper on linear dependent type theory by Peng Fu, Kohei Kishida, and Peter Selinger. The paper introduces dependent type theory in the linear and non-linear settings of the ProtoQuipper’s type system. The difficulty in defining dependent type on linear types is that the linear type can appear more than once in the typing judgment. They solve this problem by interpreting dependent types as types that depend on the shape of types which is a classical abstraction of the type. Categorical semantics of linear dependent type system is obtained as a generalization of the categorical semantics of ProtoQuipper-M [55] and is based on the state-parameter fibration.

Dependent type theory is an expressive type theory whose types correspond to formulae of first-order logic. Many properties can be represented as a first-order theory, like the reflexivity, symmetry, and transitivity of an equivalence relation can be represented as a first-order formula. The expressivity of first-order logic comes from the universal and existential quantifier, whose intended meanings are for all and there exists, respectively. The first-order universal and existential quantifiers translate into dependent product type ($\prod_{x:X} A[x]$) and dependent sum type ($\sum_{x:X} A[x]$) in the dependent type theory. Intended meanings of these dependent types are a function that transforms a proof x of type X into a proof of type $A[x]$ and the product which is equipped with two projections that give a proof of type X and a proof of type $A[x]$. In addition, it might be worth mentioning that the dependent product type $\prod_{x:X} A[x]$ may look similar to the fixpoint type $\mu X.A$ but its meaning is more similar to the meaning of a function than a fixpoint over the substitution of X by $\mu X.A$ in A .

Categorical model of dependent type theory is defined by locally cartesian closed category (LCCC), i.e. a category \mathcal{C} whose slice categories \mathcal{C}/X for object $X \in \text{Obj}(\mathcal{C})$ are all cartesian closed. In an LCCC \mathcal{C} , each type A in the type theory is interpreted as an object in a slice category \mathcal{C}/X where X represents the types of the dependent variables in A .

Dependent products and sum types are interpreted in LCCC with the help of a left and a right adjoints of a functor $f^* : \mathcal{C}/Y \rightarrow \mathcal{C}/X$ called change of base defined for each morphism $f : X \rightarrow Y$. The right adjoint is called dependent product functor $\prod_f : \mathcal{C}/X \rightarrow \mathcal{C}/Y$ and the left adjoint is called dependent sum functor $\sum_f : \mathcal{C}/X \rightarrow \mathcal{C}/Y$. Assuming that the slice categories are small, the adjunction implies the following isomorphisms between the hom sets.

$$(\mathcal{C}/X)(f^*(a), b) \cong (\mathcal{C}/Y)(a, \prod_f(b)), \quad (\mathcal{C}/X)(b, f^*(a)) \cong (\mathcal{C}/Y)(\sum_f(b), a) \quad (2.11)$$

for any $a : A \rightarrow Y$ and $b : B \rightarrow X$ which are objects of the slice categories.

To illustrate what it means, let us compare these isomorphisms with the

logical rules of quantifiers in first-order logic. Given the interpretation rules: which interpret the dependent product type $\prod_{x:X} B[x]$ as the object $\prod_f b$ in \mathcal{C}/Y and the dependent sum type $\sum_{x:X} B[x]$ as the object $\sum_f b$ in \mathcal{C}/Y where the types X and $B[x]$ are interpreted as $f : X \rightarrow Y$ and $b : B \rightarrow X$, respectively, the following intuitive equivalence relations between statements (where each proof of the statement on one side can be transformed into a proof of the statement on the other side) can be obtained from the isomorphism between the homset in Eq. (2.11):

$$A[f(x)] \vdash B[x] \iff A[y] \vdash \prod_{x:X[y]} B[x], \quad B[x] \vdash A[f(x)] \iff \sum_{x:X[y]} B[x] \vdash A[y]$$

where the entailment \vdash represents the morphism in the slice category and $f(x)$ is a type corresponding to the morphism $f : X \rightarrow Y$. In the case when $f(x) = y$, this relation could be related to the logical rules for \forall and \exists quantifiers, which are represented as follows:

$$A[y] \vdash_{x,y} B[x] \text{ iff } A[y] \vdash_y \forall_{x:X[y]} B[x], \quad B[x] \vdash_{x,y} A[y] \text{ iff } \exists_{x:X[y]} B[x] \vdash_y A[y]$$

Continuing the explanation of the interpretation, a type judgment of a term $\Gamma \vdash M : A$ is interpreted as a morphism in a slice category \mathcal{C}/X where X refers to the interpretation of the type of dependent variables. Recall that a morphism $f' : f_A \rightarrow f_B$ in a slice category \mathcal{C}/X is a morphism $f : A \rightarrow B$ from the original category which satisfies the following commute diagram, which represents a program which is modeled by the category \mathcal{C} (assuming that \mathcal{C} is a model of computation).

$$\begin{array}{ccc} A & \xrightarrow{f'=f} & B \\ & \searrow f_A & \swarrow f_B \\ & & X \end{array}$$

Now, in the model of quantum computation, a program is represented as a morphism between the objects in a monoidal category which correspond to the context and output type. Therefore, to introduce dependent types, one needs to construct a monoidal category whose slice categories have the structures mentioned above (i.e., the exponential and product types, which are adjoint and the left and the right adjunctions for each base change functor). However, the linear dependent type would require a new intuitive comprehension of the type system, which is not clear how we can do it. For example, as illustrated in [24], let f be an element of the dependent type $\prod_{a:A} B[a]$ where A is a linear type. Then, the application $f(a) : B[a]$ of f to a variable $a : A$ contains two a in the term and in the type.

Instead, the linear dependent type theory in [24] restricts the dependent types to the types dependent on classical data or set. As shown in the

ProtoQuipper-M, a program is represented as a family of morphisms in a monoidal closed category indexed by a set in the category of sets (in other words, function from a set to morphisms in the monoidal closed category). Furthermore, in its categorical semantics from Benton's linear category, the cartesian closed category (duplicable types) can be considered part of the monoidal closed category (linear types), which has a commutative comonoid structure. Therefore, to define a categorical semantics of dependent type theory with linear and non-linear types, one needs Benton's linear category (let it be an adjunction $(\mathcal{C}, \mathcal{L}, F, G)$ between the cartesian closed category \mathcal{C} and the monoidal closed category \mathcal{L}) where the slice category in \mathcal{L} over the classical objects $F(X)$ for $X \in \text{Obj}(\mathcal{C})$ is monoidal closed and has the left and the right adjunction of the base change functors.

3 - Type system and operational semantics of Proto-Quipper-L

In programming language for quantum computation, one needs to ensure that quantum data is not duplicated or deleted according to the no-cloning theorem. This has led to introducing linear types for quantum data in several quantum programming languages like quantum λ -calculus . Moreover, for quantum circuit description languages like Quipper, QWire, and Qbricks—which were introduced to allow circuit-level operations like print, reverse, and gate-count—it is necessary to assign types to quantum circuits. These new ingredients necessitate new type constants and type derivation rules in the type system. Furthermore, the meaning of these new operators regarding circuits can be formalized within the operational semantics, which is composed of configurations and reduction rules over the configuration.

The standard way of introducing linear type, like qubit, is based on linear logic. In the multiplicative part of linear logic, each proposition, which corresponds to type, is treated linearly. Then, the type qubit is introduced as a constant to the logic, implying that the quantum data is not duplicated nor erased within a well-typed term. However, there is still the necessity of classical data types like bit in the programming languages, which allows us to represent classical computation. For example, as in the teleportation example from the previous chapter, the part B applies different unitary operators depending on the classical bit. For this reason, the type system for a quantum programming language is usually incorporated with the exponential modality $!$ of linear logic, whose interpretation is a label for the duplicable types.

Moreover, in this context, we are allowed to introduce constant types for quantum circuits $\text{Circ}(A, B)$ the same way we add constants for propositions in the propositional logic. As in various versions of Proto-Quipper , the type of quantum circuits is dependent on the input and output forms of the circuit. It means that we introduce a new type of quantum circuit for each input and output form. Various operators regarding quantum circuits can then be defined as typed terms. For example, **box** and **unbox** operators are two essential operators for the construction of circuits in Quipper: **box** encapsulates a function that transforms quantum resource and makes it into a circuit object while **unbox** transforms a circuit object into a function which transmits the circuit object to the co-processor and links it with the input wires. **box** and **unbox** operators are, thus, terms of linear function type from a linear function over quantum data ($A \multimap B$) to a quantum circuit type ($\text{Circ}(A, B)$) and linear function type from a quantum circuit type ($\text{Circ}(A, B)$) to a linear function ($A \multimap B$), respectively.

Although the language and type system provides some information on terms, they do not give the exact meaning. For instance, the transmission of the circuit object by `unbox` operator appears neither in the term nor in its type. One way of formalizing its meaning is by giving an operational semantics which explains how the term reduces and changes the state of an abstract machine. In quantum circuit description languages like Quipper, these states are defined by a pair of the circuit object transmitted to the co-processor and a term. Depending on the term, the abstract machine transforms the program states according to the reduction rules. From the logical perspective given by the Curry-Howard isomorphism, a term represents a type derivation. Hence, in this sense, reducing a term in the operational semantics indeed means the normalization of the type derivation.

Several versions of Proto-Quipper and QWire have formal semantics for quantum circuit description language based on type systems and operational semantics, in a sense described above. However, they do not consider the non-deterministic computation from the interaction between the classical host and the quantum co-processor. For example, the classical data generated by measurement in the co-processor can be delivered to the host in terms of dynamic lifting, and different circuit descriptions can be delivered to the co-processor by the host depending on the measurement result. In this chapter, we explain how we extend the language and type system of Proto-Quipper to formalize dynamic lifting, or measurement, by giving operational semantics. In terms of the description of a quantum process, measurement is just an algebraic notion, and we define the notion of the quantum channel, which is extended from the quantum circuit with measurement, as algebraic structure . In order to formulate dynamic lifting, or transmission of measurement to the co-processor by `unbox`, inside a term, one needs to be able to represent a branching term whose structure corresponds to the abstract structure of the quantum channel.

3.1 . Syntax of the language

As explained above, we need a language capable of representing a non-deterministic computation introduced by dynamic lifting. A branching term represents a non-deterministic computation tree, where each leaf is a non-deterministic computation represented by a branching term. In terms of logic, branching terms corresponds to a collection of proofs where each proof represents the derivation of the sequent for the type judgment. The shape of a branching term has a close relationship with the shape of a quantum channel: on the one hand, `unbox` operator applied on a quantum channel which includes measurement, will create a branching term depending on input wires; and, on the other hand, each quantum channel constant contains a branching term

which has the same shape as the quantum channel object.

In particular, each quantum channel constant consists of input wires, a quantum channel object, and a term of the language whose shape matches the shape of the quantum channel object. A quantum channel object is an instance of an algebraic structure of a quantum channel. This algebraic structure provides a way to represent quantum channels in terms of parameterized operators. From the logical viewpoint, all quantum channel objects of the same input and output types are syntactically distinctive proofs, although they form equivalence classes over the equality based on the model of quantum computation like C^* -algebra or an algebraic theory on operators.

3.1.1 . Algebraic structure of quantum channel

In this section, we define the algebraic structure of the quantum channel with several operations related to it and the quantum channel constant, which will serve as the term for the programming language in the following. Before doing so, let us introduce the notion of quantum data as pattern and pattern type. A quantum channel can be considered as a diagram consisting of boxes, which represent quantum gates (unitary gates, free and initialization, and measurement, and wires), which connect the boxes. The wire in a diagram represents a quantum state space (analogous to C^* -algebra), which is represented by an ensemble of qubits called the pattern of the qubit. The notion of the pattern (in Definition 3.1.1) describes the way how data of a particular type is composed and controlled. Note that the definition depends on the thing that the pattern is composed of.

Definition 3.1.1 (Pattern of something).

$$\text{(Pattern)} \quad p, p_a, p_b ::= * \mid w \mid \langle p_a, p_b \rangle$$

where w refers to the something that particular pattern depends on and $\langle p_a, p_b \rangle$ represents a pair consisting of patterns p_a and p_b . For example, for the pattern of wire, w in the definition should be a wire.

There is a type of qubit in a programming language, and quantum data is represented as a variable of type qubit. We assume that there is an infinite denumerable set of variables, or names for wire, as $W = w, w_1, w_2, \dots$. Then, the pattern of the qubit is represented as a pattern of variables of type qubit within the language. To formulate it formally, we first define a particular instance of pattern for quantum types as in Definition 3.1.2. Then, from the relation of the pattern type and the pattern of variable in Definition 3.1.3, the pattern qubit p can be characterized by $p \vDash P$ derivable by definition for some pattern type P .

Definition 3.1.2 (Type of pattern).

$$\text{(PType)} \quad P, P_a, P_b ::= I \mid \mathbf{qubit} \mid P_a \otimes P_b$$

Definition 3.1.3 (Pattern type relation of the patter of variable \models).

$$\frac{}{* \models I} \quad \frac{}{w \models \mathbf{qubit}} \quad \frac{p_a \models P_a \quad p_b \models P_b}{\langle p_a, p_b \rangle \models P_a \otimes P_b}$$

Next, we define the algebraic structure of the quantum channel. Similar to the signature of quantum computation in the paper of Sam Staton [70], we define an algebraic structure of quantum channel as in Definition 3.1.4. Each quantum channel object has the form among $\epsilon(V)$, an empty quantum channel with qubits in V ; $U(\vec{V}) Q$, application of a unitary operator U over the qubits V , passed through the rest of the quantum channel Q ; $\text{init } b \ v \ Q$, allocation of a qubit of name v initialized by the boolean value b , passed through the quantum channel Q ; $\text{meas } v \ Q_1 \ Q_2$, measurement of the qubit v passed through the quantum channel either Q_1 or Q_2 depending on the measurement result; and $\text{free } v \ Q$, deallocation of a qubit v , passed through the quantum channel Q . Note that the qubit v is not discarded in the measurement case, and to discard a qubit, one needs to free the qubit explicitly.

Definition 3.1.4 (Algebraic structure of quantum computation).

(Quantum channel)

$$Q, Q_1, Q_2 ::= \epsilon(V) \mid U(\vec{V}) Q \mid \text{init } b \ v \ Q \mid \text{meas } v \ Q_1 \ Q_2 \mid \text{free } v \ Q$$

where v, b, V and \vec{V} respectively refer to variables, booleans, finite sets of variables, and a list of variables (whose element is unique in the list). A list whose element is unique is sometimes considered a set in the sequel.

Although Definition 3.1.4 represents all quantum channels that we need (which will become more apparent when we compare quantum channel and the operator algebra from Sam Staton), the definition, without any conditions on variables, generates erroneous quantum channel objects. For example, $H[v_1] \epsilon(\emptyset)$ is not a good quantum channel since the qubit v_1 , to which the Hadamard gate H is applied, is disappeared in the output. To avoid malformed quantum channels in general, we define valid quantum channels based on the state of the quantum channel. The state of a quantum channel is defined as a ternary relation on a quantum channel object, a set of variables, and a branching set of variables whose derivation rules are found in Definition 3.1.5. Here, a branching set of variables means a pattern of variables.

Definition 3.1.5 (State of quantum channel).

$$\frac{}{\mathbf{st}(\epsilon(V), V, V)} \quad \frac{\vec{V}_U \subseteq V \quad \mathbf{st}(Q, V, c)}{\mathbf{st}(U(\vec{V}_U) Q, V, c)} \quad \frac{v \notin V \quad \mathbf{st}(Q, V \cup \{v\}, c)}{\mathbf{st}(\text{init } b \ v \ Q, V, c)}$$

$$\frac{v \in V \quad \mathbf{st}(Q_1, V, c_a) \quad \mathbf{st}(Q_2, V, c_b)}{\mathbf{st}(\text{meas } v \ Q_1 \ Q_2, V, [c_a, c_b])} \quad \frac{v \in V \quad \mathbf{st}(Q, V \setminus \{v\}, c)}{\mathbf{st}(\text{free } v \ Q, V, c)}$$

We say that a quantum channel Q is valid (i.e. $\text{valid}(Q)$) if $\text{st}(Q, V_i, c_o)$ is derivable for some set of variables V_i and some pattern of the set of variables c_o . Furthermore, given the derivation of $\text{st}(Q, V_i, c_o)$, we call the set of variables V_i the input of Q (i.e. $\text{in}(Q)$) and the pattern of the set of variables c_o the output of Q (i.e. $\text{out}(Q)$). In addition, we let $\text{all}(Q)$ be the set of all variables that appear in the quantum channel object Q .

The state of a quantum channel is a partial function that maps a valid quantum channel object to the pair $(\text{in}(Q), \text{out}(Q))$, in other words, there is a unique derivation of state for the valid quantum channel. Every empty quantum channel object $\epsilon(V)$ is valid and mapped to the pair (V, V) . The application of unitary operation \overrightarrow{V}_U , assuming that Q is mapped to the pair (V, c) , is mapped to the pair (V, c) whenever the variables to which it is applied the unitary operation is a subset of the input V . Similarly, assuming that Q is mapped to the pair $(V \cup \{v\}, c)$, the allocation of a qubit $\text{init } b \ v \ Q$ is mapped to (V, c) whenever the new variable v does not already exist in the input V . A deallocation free $v \ Q$ is mapped to the pair (V, c) whenever the variable to be deallocated v does exist in the input V and Q is mapped to $(V \setminus \{v\}, c)$. Lastly, the measurement $\text{meas } v \ Q_1 \ Q_2$ is mapped to the pair $(V, [c_a, c_b])$ where Q_1 and Q_2 are mapped to (V, c_a) and (V, c_b) , respectively, and the measured qubit v is in the input V .

On top of the validity of quantum channel objects, we would like to describe a relation between quantum channel objects since some quantum channels have the same structure. In specific, two quantum channel objects have the same structure if they become syntactically exact after removing all variables in both of them. However, two quantum channels with the same structure can primarily be different by applying the operators to different variables or sets of variables. We give a precise meaning of equality of two quantum channels up to a bijection between the input variables of the quantum channels in Definition 3.1.6. Note that the equivalence of quantum channels implies that they have the same size, or cardinality, as the input variables.

Definition 3.1.6 (Equality of the quantum channel, \sim_f). Two quantum channel objects Q_1 and Q_2 are equal up to a bijection function f whenever $Q_1 \sim_f Q_2$ can be derived by the following rules.

$$\begin{array}{c}
\frac{}{\epsilon(f(V)) \sim_f \epsilon(V)} \quad \frac{Q_1 \sim_f Q_2}{U(f(\overrightarrow{V})) Q_1 \sim_f U(\overrightarrow{V}) Q_2} \\
\frac{Q_1 \sim_{f \cup \{v_2 \mapsto v_1\}} Q_2 \quad (v_1, -), (-, v_2) \notin f}{\text{init } b \ v_1 \ Q_1 \sim_f \text{init } b \ v_2 \ Q_2} \\
\frac{Q_1 \sim_{f \setminus \{v \mapsto f(v)\}} Q_2}{\text{free } f(v) \ Q_1 \sim_f \text{free } v \ Q_2} \quad \frac{Q_1 \sim_f Q_3 \quad Q_2 \sim_f Q_4}{\text{meas } f(v) \ Q_1 \ Q_2 \sim_f \text{meas } v \ Q_3 \ Q_4}
\end{array}$$

where f in the relation $Q_1 \sim_f Q_2$ refers to a bijective function from the input of Q_1 to the input of Q_2 . In particular, we say that quantum channel objects Q_1 and Q_2 are equal whenever $Q_1 \sim_{\text{id}} Q_2$.

Two empty quantum channel objects are equal up to a bijection f whenever the variables of one of them are obtained by applying f to the variables of the other. For the unitary operator application, the list of variables applied to the unitary operator in one of the quantum channel objects needs to be the point-wise application of the bijection to the variables applied to the same unitary operation in the other quantum channel object. Next, for the allocation of a new qubit, each quantum channel object can name the new qubit by a variable that does not appear in the input of the remaining part, while the remaining parts of both quantum channel objects must be equal up to a bijection extended with the pair of the variables for the new qubit. For the deallocation of a qubit, the discarded qubit must be the same up to the bijection, and the remaining parts of both quantum channel objects must be equal up to the bijection reduced by the pair of the variables of the discarded qubit. Finally, for the measurement, the measured qubit needs to be equal up to the bijection, and both remaining parts in the corresponding position from both quantum channel objects must be equal up to the bijection.

One can notice that the equality over a bijection is based on the equivalence over the renaming of new qubits in the operation of a qubit allocation. A concrete example of equal quantum channels is given by variable renaming operation in Definition 3.1.7. The renaming operation applies a renaming function σ , a bijection over the set of all variables, to each variable in a quantum channel object. Note that not all pairs of equal quantum channel objects can be represented as a pair of a quantum channel object and a renaming operation applied to it since the same variable can be used for two qubit allocations separated by a deallocation of the qubit (e.g. $\text{init tt } v \text{ (free } v \text{ (init tt } v \text{ } \epsilon(\{v\})))} \sim \text{init tt } v \text{ (free } v \text{ (init tt } u \text{ } \epsilon(\{u\})))$). However, conversely, lemma 3.1.2 shows that the quantum channel object obtained by renaming is equal to the original quantum channel object up to the renaming function over the input variables.

Definition 3.1.7 (Renaming variables in quantum channel object). Given a renaming function σ , which is a bijection over the set of all variables, we define the renaming of quantum channel as follows:

$$\begin{aligned} \sigma(\epsilon(W)) &= \epsilon(\sigma(W)) \\ \sigma(U(\vec{V}) Q) &= U(\sigma(\vec{V})) \sigma(Q) \\ \sigma(\text{init } b \text{ } v \text{ } Q) &= \text{init } b \text{ } \sigma(v) \sigma(Q) \\ \sigma(\text{free } v \text{ } Q) &= \text{free } \sigma(v) \sigma(Q) \\ \sigma(\text{meas } v \text{ } Q_1 \text{ } Q_2) &= \text{meas } \sigma(v) \sigma(Q_1) \sigma(Q_2). \end{aligned}$$

Lemma 3.1.1. *The followings hold: for any quantum channel object Q and any renaming function σ , $\mathbf{st}(Q, V, c_o)$ iff $\mathbf{st}(\sigma(Q), \sigma(V), \sigma(c_o))$ and, hence, $\mathbf{out}(\sigma(Q))_i = \sigma(\mathbf{out}(Q)_i)$ since $\sigma(c_o)_i = \sigma((c_o)_i)$.*

Proof. It can be shown by the induction on Q .

If $Q = \epsilon(V)$, then $\mathbf{st}(\epsilon(V), V, V)$ and $\mathbf{st}(\epsilon(\sigma(V)), \sigma(V), \sigma(V))$ are the unique states related the quantum channel objects $\epsilon(V)$ and $\epsilon(\sigma(V))$. Therefore, we can derive that $\mathbf{st}(\epsilon(V), V, V)$ iff $\mathbf{st}(\epsilon(\sigma(V)), \sigma(V), \sigma(V))$.

If $Q = U(\overrightarrow{V_U}) Q'$, then by induction hypothesis, we have that $\mathbf{st}(Q', V, c_o)$ iff $\mathbf{st}(\sigma(Q'), \sigma(V), \sigma(c_o))$ for any set V and any tree of sets c_o . Moreover, $\overrightarrow{V_U} \subseteq V$ iff $\sigma(\overrightarrow{V_U}) \subseteq \sigma(V)$ since σ is a bijection. Therefore, we can derive that $\mathbf{st}(U(\overrightarrow{V_U}) Q', V, c_o)$ iff $\mathbf{st}(U(\sigma(\overrightarrow{V_U})) \sigma(Q'), \sigma(V), \sigma(c_o))$.

If $Q = \text{init } b \ v \ Q'$, then by induction hypothesis, we have that $\mathbf{st}(Q', V', c_o)$ iff $\mathbf{st}(\sigma(Q'), \sigma(V'), \sigma(c_o))$ for any set V' and any tree of sets c_o . Note that $v \in V'$ iff $\sigma(v) \in \sigma(V')$ and $v \notin V'$ iff $\sigma(v) \notin \sigma(V')$ for any set V' . Therefore, we can conclude that $\mathbf{st}(\text{init } b \ v \ Q', V' \setminus \{v\}, c_o)$ iff $\mathbf{st}(\text{init } b \ \sigma(v) \ \sigma(Q'), \sigma(V'), \sigma(c_o))$ either $v \in V'$ or not.

If $Q = \text{meas } v \ Q_1 \ Q_2$, then by induction hypotheses, we have that $\mathbf{st}(Q_1, V, c_a)$ iff $\mathbf{st}(\sigma(Q_1), \sigma(V), \sigma(c_a))$ and that $\mathbf{st}(Q_2, V, c_b)$ iff $\mathbf{st}(\sigma(Q_2), \sigma(V), \sigma(c_b))$. Moreover, we have that $v \in V$ iff $\sigma(v) \in \sigma(V)$ and that $\sigma([c_a, c_b]) = [\sigma(c_a), \sigma(c_b)]$. Therefore, we can derive that $\mathbf{st}(\text{meas } v \ Q_1 \ Q_2, V, [c_a, c_b])$ iff $\mathbf{st}(\text{meas } \sigma(v) \ \sigma(Q_1) \ \sigma(Q_2), \sigma(V), \sigma([c_a, c_b]))$.

If $Q = \text{free } v \ Q'$, then by induction hypothesis, we have that $\mathbf{st}(Q', V', c_o)$ iff $\mathbf{st}(\sigma(Q'), \sigma(V'), \sigma(c_o))$. Since σ is a bijection, we hvae that $v \in V$ iff $\sigma(v) \in \sigma(V)$ for any set V . Therefore, we can derive that $\mathbf{st}(\text{free } v \ Q', V' \cup \{v\}, c_o)$ iff $\mathbf{st}(\text{free } \sigma(v) \ \sigma(Q'), \sigma(V' \cup \{v\}), \sigma(c_o))$ since $\sigma(V' \cup \{v\}) = \sigma(V') \cup \{\sigma(v)\}$. \square

Lemma 3.1.2. *For any renaming function σ and a valid quantum channel object Q , $\sigma(Q) \sim_{\sigma|in(Q)} Q$.*

Proof. Since Q is a valid quantum channel object, $\mathbf{st}(Q, V, c_o)$ is derivable for some V and c_o . We use the induction on the derivation of $\mathbf{st}(Q, V, c_o)$.

If $Q = \epsilon(V)$, then it follows that $\sigma(Q) = \epsilon(\sigma(V))$ and, hence, $\sigma(Q) \sim_{\sigma|V} Q$.

If $Q = U(\overrightarrow{V_U}) Q'$, then $\sigma(Q) = U(\sigma(\overrightarrow{V_U})) \sigma(Q')$. By induction hypothesis, we have that $\sigma(Q') \sim_{\sigma|V} Q'$. Therefore, we can derive that $\sigma(Q) \sim_{\sigma|V} Q$.

If $Q = \text{init } b \ v \ Q'$, then $\sigma(Q) = \text{init } b \ \sigma(v) \ \sigma(Q')$. By induction hypothesis, we have that $\sigma(Q') \sim_{\sigma|(V \cup \{v\})} Q'$ and $v \notin V$. Therefore, we can derive that $\sigma(Q) \sim_{\sigma|V} Q$.

If $Q = \text{meas } v \ Q_1 \ Q_2$, then $\sigma(Q) = \text{meas } \sigma(v) \ \sigma(Q_1) \ \sigma(Q_2)$. By induction hypotheses, we have that $\sigma(Q_1) \sim_{\sigma|V} Q_1$ and that $\sigma(Q_2) \sim_{\sigma|V} Q_2$. Therefore, we can derive that $\sigma(Q) \sim_{\sigma|V} Q$.

If $Q = \text{free } v \ Q'$, then $\sigma(Q) = \text{free } \sigma(v) \ \sigma(Q')$. By induction hypothesis, we hvae that $\sigma(Q') \sim_{\sigma|(V \setminus \{v\})} Q'$ and $v \in V$. Therefore, we can derive that $\sigma(Q) \sim_{\sigma|V} Q$. \square

Comparison of quantum channel with the algebraic structure of quantum computation by Sam Staton

The quantum channel in Definition 3.1.4 has a nice relationship with the first-order algebraic structure of quantum computation by Sam Staton [70]. Specifically, there are translation functions from one to the other in both directions. First, there is a map from the quantum channel object to the algebraic structure in Definition 2.10.2. Definition 3.1.8 shows how to transform a quantum channel object into an operator from the algebraic structure.

Definition 3.1.8 (Transformation rules from the quantum channel to the operator algebra of Sam Staton).

$$\begin{aligned}
t_{\epsilon(V)} &= . \\
t_{U(V) Q} &= \mathbf{apply}_U(\vec{V}, \vec{V}.t_Q) \\
t_{\text{init tt } v Q} &= \mathbf{new}(v.t_Q) \\
t_{\text{init ff } v Q} &= \mathbf{new}(v.(\mathbf{apply}_X(v, v.t_Q))) \\
t_{\text{meas } v Q_1 Q_2} &= \mathbf{meas}(v, (\mathbf{new}(v.t_{Q_1})), (\mathbf{new}(v.(\mathbf{apply}_X(v.t_{Q_2})))))) \\
t_{\text{free } v Q} &= \mathbf{meas}(v, Q, Q)
\end{aligned}$$

where t_Q , for Q being a quantum channel object, represents the operator transformed from Q .

It is straightforward to check that the transformation rules applied to a valid quantum channel Q satisfy that $\emptyset \mid \text{in}(Q) \vdash t_Q$. Next, for the other direction, the following transformation rules in Definition 3.1.9 generate a quantum channel object from an operator from Definition 2.10.2.

Definition 3.1.9 (Transformation rules from the operator algebra of Sam Staton to the quantum channel).

$$\begin{aligned}
Q_{\emptyset \mid \Delta \vdash .} &= \epsilon(\Delta) \\
Q_{\emptyset \mid \Delta, a_1, \dots, a_n \vdash \mathbf{apply}_U(\vec{a}, \vec{b}.t)} &= U(a) \sigma(Q_{\emptyset \mid \Delta, b_1, \dots, b_n \vdash t}, b, a) \\
Q_{\emptyset \mid \Delta \vdash \mathbf{new}(a.t)} &= \text{init tt } a Q_{\emptyset \mid \Delta, a \vdash t} \\
Q_{\emptyset \mid \Delta, a \vdash \mathbf{meas}(a, t, u)} &= \mathbf{meas } a (\text{free } a Q_{\emptyset \mid \Delta \vdash t}) (\text{free } a Q_{\emptyset \mid \Delta \vdash u})
\end{aligned}$$

where a and b are lists of qubit names of size n ; $\sigma(Q_t, b, a)$ refers to the quantum channel obtained from Q_t by substituting the qubit names in b by the qubit names a in the corresponding position in the vector; and Q_t represents the quantum channel for the operators t .

It is straightforward to check that for the operator term $\emptyset \mid \Delta \vdash t$, we can derive $\text{st}(Q_{\emptyset \mid \Delta \vdash t}, \Delta, c_o)$ for some tree of sets c_o . Note that the quantum channel object in the transformation rule of \mathbf{apply} contains a renaming

operator which turns the variables in b to the variables in a since renaming variables after the application is not allowed in the quantum channel. This implies that different terms in the operator algebra may be transformed into the same quantum channel.

Based on the translation, the equation theory proposed by Sam Staton 2.10.4 can be translated into an equation theory of quantum channel objects (Definition 3.1.10) based on the above translation.

Definition 3.1.10 (Algebraic theory of quantum channels). The axioms in Definition 2.10.4 is translated into the following list of equations.

1. $X(a)(\text{meas } a (\text{free } a Q_1) (\text{free } a Q_2)) = \text{meas } a (\text{free } a Q_2) (\text{free } a Q_1)$
2. $\text{meas } a (\text{free } a (U(X) Q_1)) (\text{free } a (V(X) Q_2))$
 $= D(U, V)(a :: X) (\text{meas } a (\text{free } a Q_1) (\text{free } a Q_2))$
3. $U(X) \text{free}_n X Q = \text{free}_n X Q$
4. $\text{init tt } a (\text{meas } a (\text{free } a Q_1) (\text{free } a Q_2)) = Q_1$
5. $\text{init tt } a (D(U, V)(a :: X) Q) = U(X) (\text{init tt } a Q)$
6. $\text{swap}(a :: b) Q = \sigma(Q, a :: b, b :: a)$
7. $I(V) Q = Q$
8. $(UV)(X) Q = U(X) V(X) Q$
9. $(U \otimes V)(X + +Y) Q = U(X) V(Y) Q$

Extension operation of quantum channel

We now introduce the extension operation of the quantum channel object. It introduces new variables to the input and output of a quantum channel object (in Definition 3.1.11). Extend operation is used in the operational semantics of the language in Section 3.3. Specifically, it provides a way to define the covariant composition of multiple terms in the structural reduction rules.

Based on the relationship with the operator algebra of Sam Staton, each quantum channel object is a morphism in the category of C^* -algebras. This category is equipped with two binary operators \otimes , which is a tensor product of the spaces of two C^* -algebras, and $+$, which is the coproduct representing the non-deterministic choice of operators. In terms of category theory, each operator forms a bifunctor over the category of C^* -algebras. In this setting, extend operation corresponds to the bi-functoriality of \otimes where $\text{extend}(Q, V) = Q \otimes \text{id}_V : \text{in}(Q) \otimes V \rightarrow \text{out}(Q) \otimes V$ which satisfies:

- preservation of identity $\text{extend}(\epsilon(V_0), V) = \epsilon(V_0 \cup V)$ (where $\epsilon(V)$ is interpreted as the identity morphism over the C^* -algebra of dimension $2^{|V|}$); and

- preservation of composition $\text{extend}(f \circ g, V) = \text{extend}(f, V) \circ \text{extend}(g, V)$.

Note that the output of a quantum channel object could be a coproduct of multiple C^* -algebras, which implies that the composition of quantum channel objects may require a coproduct of quantum channel objects. Although, in full generality, the extend operator should also be defined over the coproduct of quantum channel objects, we define the extend operator for only quantum channel objects without coproduct in its input as in Definition 3.1.11.

Moreover, one can notice that the Definition 3.1.11 assumes the distributivity of \otimes over $+$ meaning that $(\sum_i A_i) \otimes B = \sum_i (A_i \otimes B)$. To explain why, the output of an extended quantum channel object $\text{extend}(f, V)$, for $F : V_0 \rightarrow \sum c_o$, has the form of $(\sum c_o) \otimes V$ while by the definition of extend creates a quantum channel with output of the form $\sum_i ((c_o)_i \otimes V)$ which is the sum of the tensor product of each space of c_o and V . In fact, once we have this structure of a quantum channel object, we can intuitively explain why we may assume such equality of quantum spaces.

Continuing the comparison with the operator algebra, a quantum channel object can be considered a 3-dimension object whose axes correspond to the Hilbert space with operator \otimes , the non-deterministic choice from the coproduct $+$ and the sequence of the operators generated by the composition of operators. In particular, the two operators \otimes and $+$ create a 2-dimensional quantum space that represents the form of input and output of a quantum channel object. In this setting, there can be various ways to define a quantum space depending on the arrangement of the operators in the composition. The distributivity gives a way to equate all these different ways of representing the same space.

Definition 3.1.11 (Addition of unused wires). We define a function **extend** taking a quantum channel and a set of wire names that adds them as unused wires to the quantum channel.

$$\begin{aligned}
\mathbf{extend}(\epsilon(V), V_t) &= \epsilon(V \cup V_t) \\
\mathbf{extend}(U(\overrightarrow{V_U}) Q_1, V_t) &= U(\overrightarrow{V_U}) \mathbf{extend}(Q_1, V_t) \\
\mathbf{extend}(\text{init } b \ v \ Q_1, V_t) &= \text{init } b \ v \ \mathbf{extend}(Q_1, V_t) \\
\mathbf{extend}(\text{free } v \ Q_1, V_t) &= \text{free } v \ \mathbf{extend}(Q_1, V_t) \\
\mathbf{extend}(\text{meas } v \ Q_1 \ Q_2, V_t) &= \text{meas } v \ (\mathbf{extend}(Q_1, V_t)) (\mathbf{extend}(Q_2, V_t))
\end{aligned}$$

Lemma 3.1.3. *The following holds: for any quantum channel object Q and any sets of variables W_1 and W_2 , $\mathbf{extend}(Q, W_1 \cup W_2) = \mathbf{extend}(\mathbf{extend}(Q, W_1), W_2)$.*

Proof. By easy structural induction on Q . □

Lemma 3.1.4. *Provided that $\mathbf{all}(Q) \cap W = \emptyset$, if Q is valid then so is $\mathbf{extend}(Q, W)$. In particular, in this case, $\mathbf{st}(Q, V, c_o)$ if and only if $\mathbf{st}(\mathbf{extend}(Q, W), V \cup W, c_o \cup$*

W), where $(c_o \cup W)$ refers to the tree of variables whose leaf is the union of the leaf of c_o and W , and $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(Q) \cup W$ and $\mathbf{shape}(Q) = \mathbf{shape}(\mathbf{extend}(Q, W))$.

Proof. First of all, note that the **extned** operation does not change the shape so it is straightforward that $\mathbf{shape}(Q) = \mathbf{shape}(\mathbf{extend}(Q, W))$. For other propositions, we use the proof by induction on Q .

- $Q = \epsilon(V)$:

First, we have that $\mathbf{extend}(\epsilon(V), W) = \epsilon(V \cup W)$. It follows that $\mathbf{st}(\epsilon(V \cup W), V \cup W, V \cup W)$ and $\mathbf{st}(\epsilon(V), V, V)$. Moreover, $\mathbf{all}(\epsilon(V)) \cup W = V \cup W = \mathbf{all}(\epsilon(V \cup W))$.

- $Q = U(\vec{V}_U) Q'$:

By definition $\mathbf{extend}(U(\vec{V}_U) Q', W) = U(\vec{V}_U) \mathbf{extend}(Q', W)$.

Note that $\mathbf{all}(Q) \cap W = \emptyset$ implies that $\mathbf{all}(Q') \cap W = \emptyset$. Therefore, by induction hypothesis, $\mathbf{st}(Q', V, c_o)$ iff $\mathbf{st}(\mathbf{extend}(Q', W), V \cup W, c_o \cup W)$ where c'_o and $\mathbf{all}(\mathbf{extend}(Q', W)) = \mathbf{all}(Q') \cup W$.

Moreover, since $\mathbf{all}(Q) \cap W = \emptyset$, $\vec{V}_U \subseteq V$ iff $\vec{V}_U \subseteq V \cup W$. Therefore, we can obtain that $\mathbf{st}(Q, V, c_o)$ iff $\mathbf{st}(\mathbf{extend}(Q, W), V \cup W, c_o \cup W)$.

In addition, since $\mathbf{all}(Q) = \mathbf{all}(Q')$ and $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(\mathbf{extend}(Q', W))$, by induction hypothesis, we can derive that $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(Q) \cup W$.

- $Q = \text{init } b \ w \ Q'$:

By definition, $\mathbf{extend}(\text{init } b \ w \ Q', W) = \text{init } b \ w \ \mathbf{extend}(Q', W)$.

Note that $\mathbf{all}(Q) \cap W = \emptyset$ implies that $\mathbf{all}(Q') \cap W = \emptyset$ since $\mathbf{all}(Q') \subseteq \mathbf{all}(Q)$. Therefore, by induction hypothesis, we have that $\mathbf{st}(Q', V', c_o)$ iff $\mathbf{st}(\mathbf{extend}(Q', W), V' \cup W, c_o \cup W)$ and $\mathbf{all}(\mathbf{extend}(Q', W)) = \mathbf{all}(Q') \cup W$.

Moreover, since $\mathbf{all}(Q) \cap W = \emptyset$, it follows that $w \notin V'$ iff $w \notin (V' \cup W)$. Therefore, it follows that $\mathbf{st}(Q, V' \setminus \{w\}, c_o)$ iff $\mathbf{st}(\mathbf{extend}(Q, W), (V' \setminus \{w\}) \cup W, c_o)$. Note that $(V' \setminus \{w\}) \cup W = V' \cup W \setminus \{w\}$.

In addition, since $\mathbf{all}(Q) = \mathbf{all}(Q') \cup \{w\}$ and $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(\mathbf{extend}(Q', W)) \cup \{w\}$, by induction hypothesis, we can derive that $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(Q) \cup W$.

- $Q = \text{free } w \ Q'$:

By definition, $\mathbf{extend}(\text{free } w \ Q', W) = \text{free } w \ \mathbf{extend}(Q', W)$.

Note that $\mathbf{all}(Q) \cap W = \emptyset$ implies that $\mathbf{all}(Q') \cap W = \emptyset$ since $\mathbf{all}(Q') \subseteq \mathbf{all}(Q)$. Therefore, by induction hypothesis, we have that $\mathbf{st}(Q', V', c_o)$

iff $\mathbf{st}(\mathbf{extend}(Q', W), V' \cup W, c_o \cup W)$ and $\mathbf{all}(\mathbf{extend}(Q', W)) = \mathbf{all}(Q') \cup W$.

Moreover, since $\mathbf{all}(Q) \cap W = \emptyset$, it follows that $w \in V'$ iff $w \in (V' \cup W)$. Therefore, it follows that $\mathbf{st}(Q, V' \cup \{w\}, c_o)$ iff $\mathbf{st}(\mathbf{extend}(Q, W), (V' \cup \{w\}) \cup W, c_o)$.

In addition, since $\mathbf{all}(Q) = \mathbf{all}(Q') \cup \{w\}$ and $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(\mathbf{extend}(Q', W)) \cup \{w\}$, by induction hypothesis, we can derive that $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(Q) \cup W$.

- $Q = \text{meas } w \ Q_1 \ Q_2$:

By definition,

$$\mathbf{extend}(\text{meas } w \ Q_1 \ Q_2, W) = \text{meas } w \ \mathbf{extend}(Q_1, W) \ \mathbf{extend}(Q_2, W).$$

Note that $\mathbf{all}(Q) \cap W = \emptyset$ implies that $\mathbf{all}(Q_1) \cap W = \mathbf{all}(Q_2) \cap W = \emptyset$ since $\mathbf{all}(Q_1), \mathbf{all}(Q_2) \subseteq \mathbf{all}(Q)$. Therefore, by induction hypothesis, we have that $\mathbf{st}(Q_1, V_1, c_a)$ iff $\mathbf{st}(\mathbf{extend}(Q_1, W), V_1 \cup W, c_a \cup W)$ (and, similarly, that $\mathbf{st}(Q_2, V_2, c_b)$ iff $\mathbf{st}(\mathbf{extend}(Q_2, W), V_2 \cup W, c_b \cup W)$) and $\mathbf{all}(\mathbf{extend}(Q_1, W)) = \mathbf{all}(Q_1) \cup W$ and $\mathbf{all}(\mathbf{extend}(Q_2, W)) = \mathbf{all}(Q_2) \cup W$.

Moreover, since $\mathbf{all}(Q) \cap W = \emptyset$, it follows that $w \in V_1$ iff $w \in (V_1 \cup W)$ and that $w \in V_2$ iff $w \in (V_2 \cup W)$. Similarly, we can derive that $V_1 = V_2 = V$ iff $V_1 \cup W = V_2 \cup W = V \cup W$, for some V . Therefore, it follows that $\mathbf{st}(Q, V, [c_a, c_b])$ iff $\mathbf{st}(\mathbf{extend}(Q, W), V \cup W, [c_a, c_b] \cup W)$. Note that $[c_a, c_b] \cup W = [c_a \cup W, c_b \cup W]$.

In addition, since $\mathbf{all}(Q) = \mathbf{all}(Q_1) \cup \mathbf{all}(Q_2) \cup \{w\}$ and

$$\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(\mathbf{extend}(Q_1, W)) \cup \mathbf{all}(\mathbf{extend}(Q_2, W)) \cup \{w\}$$

by induction hypotheses, we can derive that $\mathbf{all}(\mathbf{extend}(Q, W)) = \mathbf{all}(Q) \cup W$.

□

Quantum channel constant

Next, we define the quantum channel constant by padding the quantum channel object with a pattern, which is the tree of variables, and the term, which is defined in the following subsection (Section 3.1.2). The quantum channel constant is defined in Definition 3.1.12 as the smallest set of triple (p, Q, m) closed under the construction rule in the definition. The definition can be summarized as follows: first, the quantum channel object Q should be valid; second, the support of pattern p (meaning the set of variables appearing in p) should be equal to $\text{in}(Q)$; and, third, the tree shapes of $\text{out}(Q)$ and the

term m should match and, in each branch of the tree, all variables output of Q should appear at the same branch of the term as a free variable.

The third condition is a necessary condition for the linearity of the quantum variables while it does not limit the term from using the variables several times. The linearity of quantum variables in the term will be provided by the type system presented in Section 3.2. Moreover, the condition allows that the term contains variables (i.e., classical variables) that are not present in the output of the quantum channel object.

Definition 3.1.12 (Quantum channel constant). A quantum channel constant is defined as the triple of a pattern, a quantum channel object, and a term derived from the following construction rules.

$$\frac{\text{supp}(p) \subseteq \mathbf{FV}(M)}{\mathbf{qcc}(p, \epsilon(\text{supp}(p)), M)} \quad \frac{\text{valid}(Q') \quad \text{supp}(p') = \mathbf{in}(Q') \quad \mathbf{qcc}(p, Q, m)}{\mathbf{qcc}(p', Q', m)}$$

$$\frac{\text{valid}(\text{meas } v \ Q_1 \ Q_2) \quad \text{supp}(p') = \mathbf{in}(\text{meas } v \ Q_1 \ Q_2) \quad \mathbf{qcc}(p, Q_1, m_a) \quad \mathbf{qcc}(p, Q_2, m_b)}{\mathbf{qcc}(p, \text{meas } v \ Q_1 \ Q_2, [m_a, m_b])}$$

where Q' is either $U(V) \ Q$, $\text{init } b \ v \ Q$, or $\text{free } v \ Q$ and $\text{supp}(p)$ refers to the support of pattern p . The definition of terms (i.e. M , m , m_a , and m_b) is given below.

We consider quantum channel constants *modulo* alpha-renaming: in (p, Q, m) , the pattern p is a *binding*, as are the init operations inside Q . Moreover, we ignore the predicate symbol \mathbf{qcc} of a derivable quantum channel constant and represent it as just $(-, -, -)$ unless it is necessary. Therefore, when we write (p, Q, m) in the rest of the paper, we mean: “The quantum channel constant with (p, Q, m) as a representative element”.

In quantum circuit description languages like QWire and Quipper, quantum circuit constants are first-class objects of type $\text{Circ}(-, -)$ on the side of the classical host. In the case of a quantum circuit, the patterns in the quantum circuit constant give parameters for the Circ type constructor. Similarly, we let $\text{QChan}(-, -)$ be the type for the quantum channel constants. Moreover, similar to the quantum circuit constant, the quantum channel constant is that the quantum channel constant includes a term. This imposes extra structures on quantum channel constants—quantum channel constants can be evaluated to an equivalent quantum channel constant whose term is a value. However, in contrast to the quantum circuit constant, the term in a quantum channel constant can have any type, meaning that type of quantum channel $\text{QChan}(A, B)$ can take any type B while A needs to be a pattern type of qubit.

Toward the quantum channel operators—`box` and `unbox`

Lastly, we introduce quantum channel operators for the construction of quantum channel objects and the lifting—`box` and `unbox`—which will be formally

defined later in the thesis. Box operator translates a program into a corresponding quantum channel constant. This operation is based on the interpretation of the program as a quantum channel, where the program creates a quantum channel while being evaluated. In specific, all functions from a pattern of the qubit to any type have a corresponding quantum channel constant. The type system guarantees that the box operator is applied to functions that satisfy the constraint.

Then, unbox operator is to buffer a quantum channel constant to the channel from the classical host to the quantum co-processor. Remembering that a quantum channel constant consists of a pattern of the qubit, quantum channel object, and a term, unbox operator evaluates the term for a given instance of the pattern while extending the quantum channel object.

In fact, box and unbox can be considered as maps between the terms and quantum channel constants. However, since the definitions of quantum channel constant and term depend on each other, one needs to distinguish the depth of quantum channel constant (i.e., qcd), which is defined formally in Section 3.2. In this case, the box operator takes a term of the quantum channel depth n and outputs a quantum channel constant of depth $n + 1$ while unbox takes a quantum channel constant of depth $n + 1$ and outputs a term of depth n .

A natural question is whether the box and the unbox operators are inverses. Given that unbox has a side effect that pushes quantum channel objects to quantum co-processor, the box and the unbox operators are not inverse to each other except for the case where unbox has nothing to push. However, when we model the side effect in semantics, we may show that these two operators are inverse over the equivalence of the semantics .

3.1.2 . Definition of the language

Now, we define the syntax of the language Proto-Quipper-L, which is about how to construct the term in the language. Compared to the previous Proto-Quipper languages, there are two main differences in Proto-Quipper-L.

Firstly, as mentioned in the previous section, the quantum channel constant will serve as a constant term like the constant terms `*`, `tt` and `ff` that does not depend on anything. However, since a quantum channel constant includes a term inside, it can also be considered as encapsulating a term inside a constant. Thus, it is similar to the abstraction in λ -calculus while it requires special constant operators `boxP` and `unbox` for the abstraction and application. We specify this relationship in more detail in the type systems in Section 3.2.

Secondly, since the quantum channel has a non-trivial control flow, the terms at the leaf of each branch of the control flow can have different forms, although they share the same type (provided that the term is well-typed). To incorporate this branching structure in the quantum channel constant, we

extend the language with the branching constructor $([-, -])$. A term of the form $[M, N]$ represents a computation that has probabilistically branched, and that is performing either M or N . Note that the corresponding probability is not kept in the term, however.

This new structure is related to the if-then-else statement, and the if-then-else statements, the branching term, and the quantum channel constant rely on the same coproduct structure in the categorical model (Section 4). Now, the formal definition of the terms is given in Definition 3.1.13

Definition 3.1.13 (Syntax of Proto-Quipper-L).

(Term)

$$M, M_a, M_b ::= x \mid * \mid \text{tt} \mid \text{ff} \mid (p, Q, m) \mid \lambda x.M \mid M_a M_b \mid \langle M_a, M_b \rangle \mid \\ \mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b \mid \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b \mid \\ \mathbf{box}_P \mid \mathbf{unbox}$$

(Branching term)

$$m, m_a, m_b ::= M \mid [m_a, m_b]$$

The constant $*$ stands for the unit term, while tt and ff stand for the booleans true and false. The term (p, Q, m) represents the quantum channel constant— p is a structured set of input wires of the quantum channel Q , m is now a pattern of the term with the same structure with Q . The rest of the constructors of the language are standard: abstraction, application, pair, let, and conditional statements; and the quantum channel operators: \mathbf{box}_P and \mathbf{unbox} . The index P in \mathbf{box}_P stands for a pattern-type: by abuse of notation, we omit it when not needed. Finally, branching terms are constructed using terms and the branching constructor.

The terms are defined as the equivalence classes over the α -equivalence meaning that two terms are equal if one is obtained by renaming closed variables of the other. To explain it, a variable in a term is free when it is not bounded, or, more concretely, a set of free variables of a term can be defined inductively as in Definition 3.1.14. Then the closed variables in a term are defined as all variables which are no free variables.

Definition 3.1.14 (Free variables in terms). Free variables of a term are defined inductively as follows.

$$\begin{aligned} \mathbf{FV}(x) &= \{x\} & \mathbf{FV}(*) &= \mathbf{FV}(\text{tt}) = \mathbf{FV}(\text{ff}) = \mathbf{FV}(\mathbf{box}_P) = \mathbf{FV}(\mathbf{unbox}) = \emptyset \\ \mathbf{FV}(\mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b) &= \mathbf{FV}(M_a) \cup (\mathbf{FV}(M_b) \setminus \{x, y\}) & \mathbf{FV}(\lambda x.M) &= \mathbf{FV}(M) \setminus \{x\} \\ \mathbf{FV}(M_a M_b) &= \mathbf{FV}(\langle M_a, M_b \rangle) = \mathbf{FV}(M_a) \cup \mathbf{FV}(M_b) & \mathbf{FV}([m_a, m_b]) &= \mathbf{FV}(m_a) \cup \mathbf{FV}(m_b) \\ \mathbf{FV}(\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b) &= \mathbf{FV}(M) \cup \mathbf{FV}(M_a) \cup \mathbf{FV}(M_b) & \mathbf{FV}((p, Q, m)) &= \cup_i (\mathbf{FV}(m_i) \setminus \mathbf{out}(Q)_i) \end{aligned}$$

Next, given a renaming function σ and a term m , $\sigma(m)$ renames all variables in m by σ (Definition 3.1.15).

Definition 3.1.15 (Renaming variables in term).

$$\begin{aligned}
\sigma(x) &= \sigma(x) & \sigma(*) &= * & \sigma(\text{tt}) &= \text{tt} & \sigma(\text{ff}) &= \text{ff} & \sigma(\text{unbox}) &= \text{unbox} & \sigma(\text{box}_P) &= \text{box}_P \\
\sigma(\text{let } \langle x, y \rangle = M_a \text{ in } M_b) &= \text{let } \langle \sigma(x), \sigma(y) \rangle = \sigma(M_a) \text{ in } \sigma(M_b) & \sigma(\lambda x.M) &= \lambda(\sigma(x)).(\sigma(M)) \\
\sigma(M_a M_b) &= \sigma(M_a)\sigma(M_b) & \sigma(\langle M_a, M_b \rangle) &= \langle \sigma(M_a), \sigma(M_b) \rangle & \sigma([m_a, m_b]) &= [\sigma(m_a), \sigma(m_b)] \\
\sigma(\text{if } M \text{ then } M_a \text{ else } M_b) &= \text{if } \sigma(M) \text{ then } \sigma(M_a) \text{ else } \sigma(M_b) & \sigma((p, Q, m)) &= (\sigma(p), \sigma(Q), \sigma(m))
\end{aligned}$$

where $\sigma(Q)$ is defined in Definition 3.1.7.

Lemma 3.1.5. *The following holds: $\sigma(\mathbf{FV}(u_i)) = \mathbf{FV}(\sigma(u)_i)$.* \square

Next, we define value as a subset of terms in Definition 3.1.16. Values are the terms that do not induce computation in the sense that there is not any further reduction of the values in operational semantics (Lemma 3.4.6 in Section 3.3). Moreover, in terms of categorical semantics, each morphism in the Kleisli category corresponding to a value satisfies the property that it can be decomposed by a morphism in the original category and the unit of the Kleisli category (Lemme 5.3.1 in Section 5).

Definition 3.1.16 (Values of Proto-Quipper-L).

$$\begin{aligned}
\text{(Value)} \quad V, V_a, V_b &::= x \mid * \mid \text{tt} \mid \text{ff} \mid \lambda x.M \mid \langle V_a, V_b \rangle \mid \\
&\quad (p, Q, v) \mid \text{box}_P \mid \text{unbox} \mid \text{unbox}(V) \\
\text{(Branching value)} \quad v, v_a, v_b &::= V \mid [v_a, v_b]
\end{aligned}$$

The value consists of the variables, unit term, true and false boolean values, quantum channel constants (whos term is value), abstraction, pair of values, and quantum channel operators. Moreover, for non-trivial structures, values are defined constructively as the branching term of two values. We prove that the value corresponds to the term, which does not reduce by the operational semantics in the progress lemma.

The following lemma states that the value is preserved under the substitution of variables by values.

Lemma 3.1.6. *If v and V' are values, then $v[V'/x]$ is a value.*

Proof. Proof by induction on v .

- $v = x$: since V' is a value, $v[V'/x] = V'$ is a value.
- $v = y$: $v[V'/x] = v$ is a value.
- $v = *, \text{tt}, \text{ff}, \text{box}, \text{unbox}$: $v[V'/x] = v$ is value.
- $v = \lambda x.M$: $v[V'/x] = v$ is value.
- $v = \lambda y.M$: $v[V'/x] = \lambda y.M[V'/x]$ is a value.

- $v = \langle V_a, V_b \rangle$: $v[V'/x] = \langle V_a[V'/x], V_b[V'/x] \rangle$ is a value since $V_a[V'/x]$ and $V_b[V'/x]$ are value by induction hypothesis.
- $v = (p, Q, v_a)$: First, we let $(p, Q, v_a)[V'/x] = (p, Q, v_b)$. In each branch i , there are two case for substitution:
 - if $x \in \mathbf{out}(Q)_i$, then $(v_b)_i = (v_a)_i[V'/x] = (v_a)_i$, and
 - if $x \notin \mathbf{out}(Q)_i$, then $(v_b)_i = (v_a)_i[V'/x]$ is value by induction hypothesis.

Therefore, v_b and (p, Q, v_b) are values.

- $v = \mathbf{unbox}(V)$: It follows from the induction hypothesis that $V[V'/x]$ is value. Therefore, $v[V'/x] = \mathbf{unbox}(V[V'/x])$ is value.
- $v = [v_a, v_b]$: By induction hypothesis, $v_a[V'/x]$ and $v_b[V'/x]$ are values. Therefore, $v[V'/x] = [v_a[V'/x], v_b[V'/x]]$ is value.

□

Since we have a formal language, we can now represent quantum programs that generate quantum channels. Note that the constants for unitary gates, the new qubit allocation operator, and the measurement from the quantum lambda calculus [63] are missing in our syntax. As in the case of Proto-Quipper, they can be defined with the unbox operator and quantum channel constants. For instance, we can construct a measurement operation inputting a qubit and outputting a boolean (and the measured wire), graphically represented as follows:

$$\mathbf{meas} ::= \mathbf{unbox} \left(q, q \text{ --- } \boxed{\text{meas}} \begin{matrix} \text{---} q \\ \text{---} q \end{matrix}, \bullet \begin{matrix} \text{---} \langle \mathbf{tt}, q \rangle \\ \text{---} \langle \mathbf{ff}, q \rangle \end{matrix} \right). \quad (3.1)$$

The tuple consists of a singleton wire name q , the quantum channel object ($\mathbf{meas} \ q \ \epsilon \{q\} \ \epsilon \{q\}$), and the branching term $[\langle \mathbf{tt}, q \rangle, \langle \mathbf{ff}, q \rangle]$. Note that the measurement operator we wrote here returns both a qubit and a boolean: we could discard the qubit using a quantum channel constructor “free”. Similarly, we also build the macros $\mathbf{init}(b)$ and $\mathbf{free}(x)$ which respectively allocates a new qubit in state b and frees a qubit x

$$\begin{aligned} \mathbf{init}(b) & ::= \mathbf{unbox} \left(*, * \boxed{\mathbf{init} \ b \ x} x, \bullet \text{---} x \right) \\ \mathbf{free} & ::= \mathbf{unbox} \left(q, q \text{ --- } \boxed{\mathbf{free} \ q} *, \bullet \text{---} * \right). \end{aligned} \quad (3.2)$$

On top of it, we define the following unitary operators. Note that the unitary operator in the quantum channel object can be defined similarly to the operator algebra of Sam Staton [70] as explained in Section 3.1.1.

$$\begin{aligned}
X(q) &::= \text{unbox} \left(q, q \text{---} \boxed{X[q]} \text{---} q, \bullet \text{---} q \right) \\
H(q) &::= \text{unbox} \left(q, q \text{---} \boxed{H[q]} \text{---} q, \bullet \text{---} q \right) \\
Z(q) &::= \text{unbox} \left(q, q \text{---} \boxed{Z[q]} \text{---} q, \bullet \text{---} q \right) \\
\text{CNOT}(q_1, q_2) &::= \text{unbox} \left(\langle q_1, q_2 \rangle, \langle q_1, q_2 \rangle \text{---} \boxed{\text{CNOT}[q_1, q_2]} \text{---} \langle q_1, q_2 \rangle, \bullet \text{---} \langle q_1, q_2 \rangle \right)
\end{aligned} \tag{3.3}$$

Note that a variation of measurement that returns a Boolean instead of a pair of a Boolean and a qubit (e.g., measurement in Sam Staton’s algebraic language [70]) can be obtained by applying free to the qubit which has been measured. We define a quantum channel for this variation in Equation (3.4).

$$\text{meas}_f ::= \text{unbox} \left(q, q \text{---} \begin{array}{c} \text{free } q \text{---} * \\ \text{free } q \text{---} * \end{array}, \bullet \begin{array}{c} \text{tt} \\ \text{ff} \end{array} \right). \tag{3.4}$$

With these operators, we can define programs for non-trivial quantum channels as shown in the following examples (Example 3.1.1 and Example 3.1.2). Those two examples will be reappearing to illustrate the type system, operational semantics, and categorical semantics.

Example 3.1.1. *The term exp in Example 1.1 can be now represented as a term in the language.*

$$\text{let } \langle b, v_c \rangle = \text{meas}(v_c) \text{ in } \text{if } b \text{ then } \langle \text{init}(\text{tt}), \text{free}(v_c) \rangle \text{ else } \langle v_c, * \rangle$$

where the circuit constants meas , init and free are defined in the Equation 3.1 and Equation 3.2.

Example 3.1.2. *In this example, we show how to represent the quantum teleportation (which is introduced in background) in the language with the quantum channel operators in Equation 3.1, Equation 3.2, Equation 3.3, and Equation 3.4. It consists of three parts—one that creates a bell state (Bell) and the two processes for Alice (A) and Bob (B).*

First, the preparation of bell state is a function defined as follows:

$$(\text{Bell}) ::= \text{CNOT} \langle H(\text{init}(\text{tt})), \text{init}(\text{tt}) \rangle$$

Second, the processes for Alice and Bob is defined as follows:

$$(A) ::= \lambda y. \lambda x. (\text{let } \langle x, y \rangle = \text{CNOT}(x, y) \text{ in } \langle \text{meas}_f(H(x)), \text{meas}_f(y) \rangle)$$

$$(B) ::= \lambda q. \lambda xy. (\text{let } \langle x, y \rangle = xy \text{ in } U_{xy})$$

where

$$(U_{xy}) ::= \mathbf{if} \ x \ \mathbf{then} \ (\mathbf{if} \ y \ \mathbf{then} \ X(Z(q)) \ \mathbf{else} \ Z(q)) \\ \mathbf{else} \ (\mathbf{if} \ y \ \mathbf{then} \ X(q) \ \mathbf{else} \ q)$$

Then, teleportation is defined as the following term:

$$(\mathit{tel}) ::= \mathbf{let} \ \langle y, q \rangle = \mathit{Bell} \ \mathbf{in} \ \langle A(y), B(q) \rangle.$$

3.2 . Type system

Type systems are used to decide well-formed terms in a quantum programming language, for example, the terms satisfying the no-cloning property of quantum mechanics. Since quantum data (i.e., wires) are linear, meaning that a wire is used exactly once, while classical data, including boxed circuits, are non-linear, previous works [56, 55, 47, 53] use linear/non-linear type systems to encompass both quantum and classical data types. For the same reason, we define a linear/non-linear type system for Proto-Quipper-L.

The type system is defined the same way as the other linear/non-linear type systems for quantum programming language. On top of it, for the quantum channel constant, we define the type $\text{QChan}(P, A)$ which is parameterized by a pattern type P and a type A . Next, we extend typing judgment for the branching term while allowing each term in the branching term to take the same type.

Now, let us begin to introduce the type system by giving the definition type in Definition 3.2.1. Types for the language consist in constant types I , bool , qubit ; the type of quantum channels $\text{QChan}(P, A)$ with input of type P and output of type A ; the function type $A_a \multimap A_b$; the type for pairs $A_a \otimes A_b$; and the classical type $!A$. The bang constructor $!A$ indicates its instances are duplicable and deletable.

Definition 3.2.1 (Definition of type).

$$\begin{aligned} (\text{Type}) \quad A, A_a, A_b ::= & \ I \mid \text{bool} \mid \mathbf{qubit} \mid \text{QChan}(P, A) \\ & \mid A_a \multimap A_b \mid A_a \otimes A_b \mid !A \end{aligned}$$

where P refers to the type of patterns, that is, first-order types constructed from constant types and tensors.

Although quantum channels are first-class data, they can contain any term inside. Since a term includes quantum channels, a term can have a finite depth of the quantum channel, what we call the nesting depth of the QChan , which is defined in Definition 3.2.2. Then, all types can be indexed by the nesting depth of the QChan and this index is used in the induction on the type.

Definition 3.2.2 (Nesting QChan depth of type).

$$\begin{array}{c} \overline{\mathbf{qcd}(I) = \mathbf{qcd}(\text{bool}) = \mathbf{qcd}(\text{qubit}) = 0} \\ \frac{\mathbf{qcd}(A_a) = x \quad \mathbf{qcd}(A_b) = y}{\mathbf{qcd}(A_a \multimap A_b \mid A_a \otimes A_b) = \mathbf{max}(x, y)} \quad \frac{\mathbf{qcd}(A) = x}{\mathbf{qcd}(\text{QChan}(P, A)) = x + 1} \end{array}$$

where x and y are integers and $\mathbf{max}(x, y)$ takes the max of the two.

Next, since a term can contain unbounded variables, type judgment requires typing context that assigns types to the variables. Conventional typing context is a list of pairs that correspond variables to type. However, since the branching term can contain different unbounded variables, the typing context also needs to be branching. The typing context for branching and non-branching terms is formally defined in Definition 3.2.3.

Definition 3.2.3 (Typing context and branching typing context).

$$\begin{array}{ll} \text{(Non-branching typing context)} & \Gamma, \Gamma_a, \Gamma_b ::= (x_l : A_l)_{l \in L} \\ \text{(Branching typing context)} & \gamma, \gamma_a, \gamma_b ::= \Gamma \mid \gamma_a \times \gamma_b \end{array}$$

where $(x_l : A_l)_{l \in L}$ represents the list of pairs $(x_l : A_l)$ indexed by L and the product $\gamma_a \times \gamma_b$ represents the typing context for the branching term. All variables appearing in each non-branching typing context are assumed to be distinct. Also, note that each list is not ordered in the typing context.

We let the composition Γ_a, Γ_b of typing contexts Γ_a and Γ_b by comma $,$ refer to the disjoint union of typing contexts. Then, the non-branching typing context is equally represented as the composition of the pairs of a variable and a type by comma $,$ as follows:

$$\Gamma = (x_{l_1} : A_{l_1}), \dots, (x_{l_k} : A_{l_k}).$$

For convenience, we divide typing context into two parts: one consists of strictly linear data, whose types do not start with $!$, and the other contains exponential data, whose types start with $!$. In the sequel, we let Q and $!\Delta$ refer to strictly linear context and exponential context, respectively. Moreover, we consider a non-branching typing context Γ a disjoint union of strictly linear typing context Q and exponential typing context $!\Delta$, represented as $\Gamma = !\Delta, Q$. Also, for simplicity, we let $\text{TC}_Q(X)$ refer to $(x_i : \text{qubit})_{x_i \in X}$, meaning the quantum context of variables X .

Next, we define a relation called type judgment over the typing context, term, and type, as in Definition 3.2.4.

Definition 3.2.4 (Type judgement).

$$\begin{array}{ll} \text{(Type judgement)} & \Gamma \vdash M : A \\ \text{(Branching type judgement)} & \gamma \vdash m : A \end{array}$$

A type judgment is true if, and only if, it is derivable from the type inference rules. For the quantum channel constants to be valid, it needs to be checked that the quantum channel is well defined and each term in the branching term has the same type under the proper context. For this reason, we define a relation called \mathbf{vBind} over the exponential type context, the output of the quantum channel, branching terms, and type as in Definition 3.2.5. Intuitively, \mathbf{vBind} implies that, first, the output of quantum channel and branching term share the same shape; second, for each branch, the output of quantum channel should be treated as linear variables; and, third, the term in each branch admits the given type within the context consists of the exponential context and the qubits from the output of the quantum channel.

Definition 3.2.5 (Validity of binding, \mathbf{vBind}).

$$\frac{Q \cap \mathbf{FV}(!\Delta) = \emptyset \quad !\Delta, \mathbf{TC}_Q(Q) \vdash M : A}{\mathbf{vBind}(!\Delta, Q, M, A)} (\mathbf{vBind}_{nb})$$

$$\frac{\mathbf{vBind}(!\Delta, c_a, m_a, A) \quad \mathbf{vBind}(!\Delta, c_b, m_b, A)}{\mathbf{vBind}(!\Delta, [c_a, c_b], [m_a, m_b], A)} (\mathbf{vBind}_b)$$

where $\mathbf{FV}(!\Delta)$ refer to the set of free variables in M and the names of variables in the context $!\Delta$.

Next, the typing derivation rules are defined below in Definition 3.2.6.

Definition 3.2.6 (Type derivation rules).

$$\frac{}{!\Delta, (x : A) \vdash x : A} (\text{var}) \quad \frac{!\Delta, Q \vdash M : !A}{!\Delta, Q \vdash M : A} (\text{d}) \quad \frac{!\Delta \vdash V : A \quad V \text{ is value}}{!\Delta \vdash V : !A} (\text{p})$$

$$\frac{}{!\Delta \vdash * : I} (\text{l}) \quad \frac{}{!\Delta \vdash \text{tt} : \text{bool}} (\text{tt}) \quad \frac{}{!\Delta \vdash \text{ff} : \text{bool}} (\text{ff})$$

$$\frac{!\Delta, Q, (x : A_a) \vdash M : A_b}{!\Delta, Q \vdash \lambda x. M : A_a \multimap A_b} (\multimap_I) \quad \frac{!\Delta, Q_a \vdash M_a : A_a \multimap A_b \quad !\Delta, Q_b \vdash M_b : A_a}{!\Delta, Q_a, Q_b \vdash M_a M_b : A_b} (\multimap_E)$$

$$\frac{!\Delta, Q_a \vdash M_a : A_a \quad !\Delta, Q_b \vdash M_b : A_b}{!\Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b} (\otimes_I)$$

$$\frac{!\Delta, Q_a \vdash M_a : A_a \otimes A_b \quad !\Delta, Q_b, (x : A_a), (y : A_b) \vdash M_b : A}{!\Delta, Q_a, Q_b \vdash \mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b : A} (\otimes_E)$$

$$\frac{!\Delta, Q_a \vdash M : \text{bool} \quad !\Delta, Q_b \vdash M_a : A \quad !\Delta, Q_b \vdash M_b : A}{!\Delta, Q_a, Q_b \vdash \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b : A} (\text{if})$$

$$\frac{}{!\Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A)} (\text{box}) \quad \frac{}{!\Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A)} (\text{unbox})$$

$$\frac{\gamma_a \vdash m_a : A \quad \gamma_b \vdash m_b : A}{\gamma_a \times \gamma_b \vdash [m_a, m_b] : A} (\text{b}) \quad \frac{p \vDash P \quad \mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)}{!\Delta \vdash (p, Q, m) : !\text{QChan}(P, A)} (\text{QChan}_I)$$

The typing rules ensure that all terms constituting a branching term share the same type, and the quantum channel constants have types built with the constructor of type \mathbf{QChan} . For instance, the term meas in Eq. (3.1) is typed as $! \Delta \vdash \text{meas} : \mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{qubit}$ as in Example 3.2.1. Similarly, the term $\text{init}(b)$, and free , in Eq. (3.2) has type $I \multimap \mathbf{qubit}$, and $\mathbf{qubit} \multimap I$, respectively.

Example 3.2.1 (Typing derivation of quantum channel constants). *The quantum channel constants in Eq. 3.1 and Eq. 3.2 admit the following typing derivations.*

$$\frac{\frac{}{! \Delta \vdash \text{unbox} : A_m \multimap B_m} \text{(unbox)} \quad \frac{}{! \Delta \vdash \left(q, q \text{---} \begin{array}{c} \text{---} q \\ \boxed{\nearrow} \\ \text{---} q \end{array}, \bullet \left(\begin{array}{c} \langle \text{tt}, q \rangle \\ \langle \text{ff}, q \rangle \end{array} \right) \right) : A_m}{} \pi_m}{! \Delta \vdash \text{meas} : B_m} \text{(}\multimap_E\text{)}$$

where $A_m = \mathbf{QChan}(\mathbf{qubit}, \mathbf{bool} \otimes \mathbf{qubit})$ and $B_m = \mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{qubit}$ and π_m represents the following type derivation.

$$\pi_m = \frac{\frac{\frac{}{p \vDash \mathbf{qubit}}}{! \Delta \cap \{(q|q)\} = \emptyset} \quad \frac{}{! \Delta, (q|q) : \mathbf{qubit} \vdash \langle \text{tt}, (q|q) \rangle : \mathbf{bool} \otimes \mathbf{qubit}}{\mathbf{vBind}(! \Delta, \{(q|q)\}, \langle \text{tt}, (q|q) \rangle, \mathbf{bool} \otimes \mathbf{qubit})} \text{(vBind}_{nb}\text{)}}{\mathbf{vBind}(! \Delta, [\{q\}, \{q\}], [\langle \text{tt}, q \rangle, \langle \text{ff}, q \rangle], \mathbf{bool} \otimes \mathbf{qubit})} \text{(vBind}_b\text{)}} \text{(QChan}_I\text{)}}{! \Delta \vdash \left(q, q \text{---} \begin{array}{c} \text{---} q \\ \boxed{\nearrow} \\ \text{---} q \end{array}, \bullet \left(\begin{array}{c} \langle \text{tt}, q \rangle \\ \langle \text{ff}, q \rangle \end{array} \right) \right) : !A_m}$$

assuming that x and y is not contained in the exponential typing context $! \Delta$. Note that the $(A \mid B)$ represents the two sides of the typing rule (\mathbf{vBind}_b) : A refers to the variables for the left \mathbf{vBind} relation and B refers to the variables for the right.

Similarly, we can derive the following type judgements:

$$\begin{aligned} ! \Delta \vdash \text{init}(b) &: I \multimap \mathbf{qubit} \\ ! \Delta \vdash \text{free} &: \mathbf{qubit} \multimap I \\ ! \Delta \vdash \text{meas}_f &: \mathbf{qubit} \multimap \mathbf{bool} \end{aligned}$$

and

$$\begin{aligned} ! \Delta, q : \mathbf{qubit} &\vdash X(q) : \mathbf{qubit} \multimap \mathbf{qubit} \\ ! \Delta, q : \mathbf{qubit} &\vdash H(q) : \mathbf{qubit} \multimap \mathbf{qubit} \\ ! \Delta, q : \mathbf{qubit} &\vdash Z(q) : \mathbf{qubit} \multimap \mathbf{qubit} \\ ! \Delta, q : \mathbf{qubit} &\vdash \text{CNOT}(q_1, q_2) : \mathbf{qubit} \otimes \mathbf{qubit} \multimap \mathbf{qubit} \otimes \mathbf{qubit} \end{aligned}$$

Note that those type judgments are not unique type judgments for the terms. As an example, we can also obtain the type judgement $! \Delta \vdash \text{free} : \mathbf{qubit} \multimap !I$ from the fact that $! \Delta \vdash * : !I$ can be derived from $! \Delta \vdash * : I$ by the promotion rule (p).

Remembering that term represents a proof of the sequent corresponding to the type judgment in terms of logic, the constants `qubit` and `QChan(-,-)` can be considered as propositional constants whose proofs are represented by variables and by quantum channel constants, respectively. Also, note that there are types of quantum channels of different input and output shapes, i.e., types of input and output.

As mentioned in Section 3.1.2, the encapsulation is similar to the abstraction. In particular, the type derivation for `unbox($p, \epsilon(\text{supp}(p)), M$)` in Lemma 3.2.1 is similar to the typing rule \multimap_I for $\lambda p.M$. However, the premises of the type derivation takes `supp(p)` in its typing context, which deconstruct the structure of the pattern p while the rule \multimap_I for abstraction keeps the structure of the input.

Lemma 3.2.1. *Unbox can be thought of as an encapsulation of a term in a quantum channel constant. For some pattern p of type P such that $\text{supp}(p) = \{p_1, \dots, p_k\}$, we can obtain the following type derivation.*

$$\frac{!\Delta, (\mathbf{supp}(p) : \mathbf{qubit}) \vdash M : A}{!\Delta \vdash \text{unbox}(p, \epsilon(\mathbf{supp}(p)), M) : P \multimap A}$$

Proof.

$$\frac{\frac{\frac{}{!\Delta \vdash \text{unbox} : A_M \multimap B_M}(\text{unbox}) \quad \frac{}{!\Delta \vdash (p, \epsilon(\mathbf{supp}(p)), M) : A_M}(\text{---})}{!\Delta \vdash \text{unbox}(p, \epsilon(\mathbf{supp}(p)), M) : P \multimap A}(\multimap_E)}{\vdots}$$

and

$$\frac{\frac{\frac{}{p \Vdash P} \quad \frac{!\Delta, (\mathbf{supp}(p) : \mathbf{qubit}) \vdash M : A}{\mathbf{vBind}(!\Delta, \mathbf{supp}(p), M, A)}(\mathbf{vBind}_{nb})}{!\Delta \vdash (p, \epsilon(\mathbf{supp}(p)), M) : !\text{QChan}(P, A)}(\text{QChan}_I)}{\frac{}{!\Delta \vdash (p, \epsilon(\mathbf{supp}(p)), M) : A_M}(\text{d})}$$

where $A_M = \text{QChan}(P, A)$ and $B_M = P \multimap A$ for some PType P . \square

Then, by letting $\lambda * .M$ refers to `unbox($*, \epsilon(\emptyset), M$)` as in Equation 3.5, we can obtain the type derivation for the abstraction of a term M with no variable, as in Corollary 3.2.1.1.

$$\lambda * .M = \text{unbox}(*, \epsilon(\emptyset), M) \tag{3.5}$$

It allows us to define the teleportation example as a function as in Example 3.2.2.

Corollary 3.2.1.1. *Then, by Lemma 3.2.1, by letting $p = *$, we obtain the following type derivation.*

$$\frac{! \Delta \vdash M : A}{! \Delta \vdash \lambda * . M : I \multimap A}$$

Moreover, note that the typing rule (unbox) in our type system implies that the typing rule (unbox') in Eq. (3.6).

$$\frac{}{! \Delta \vdash \text{unbox} : !\text{QChan}(P, A) \multimap !(P \multimap A)} \text{(unbox')} \quad (3.6)$$

Indeed, we can obtain the typing derivation (unbox'), by defining a variation of unbox as in Eq. (3.7), as in Lemma 3.2.2.

$$\text{unbox}_{dup} ::= \lambda x. \text{unbox}(x) \quad (3.7)$$

Lemma 3.2.2. *Given that there is no quantum variables in the context, the duplicable unbox operator unbox_{dup} admits the type $!\text{QChan}(P, A) \multimap !(P \multimap A)$.*

Proof. The typing derivation for $! \Delta \vdash \text{unbox}_{dup} : !\text{QChan}(P, A) \multimap !(P \multimap A)$ is shown below. Note that $\text{unbox}(x)$ is a value since variable x is a value.

$$\frac{\begin{array}{c} \vdots \\ \hline ! \Delta, (x : !\text{QChan}(P, A)) \vdash \text{unbox}(x) : P \multimap A \quad \text{unbox}(x) \text{ is value} \end{array}}{! \Delta, (x : !\text{QChan}(P, A)) \vdash \text{unbox}(x) : !(P \multimap A)} \text{(p)}$$

$$\frac{}{! \Delta \vdash \lambda x. \text{unbox}(x) : !\text{QChan}(P, A) \multimap !(P \multimap A)} \text{(\multimap}_I)$$

and

$$\frac{\frac{}{! \Delta, (x : !\text{QChan}(P, A)) \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A)} \text{(unbox)} \quad \frac{\frac{}{! \Delta, (x : !\text{QChan}(P, A)) \vdash x : !\text{QChan}(P, A)} \text{(var)} \quad ! \Delta, (x : !\text{QChan}(P, A)) \vdash x : \text{QChan}(P, A)} \text{(d)}}{! \Delta, (x : !\text{QChan}(P, A)) \vdash \text{unbox}(x) : P \multimap A} \text{(\multimap}_E)}$$

□

Example 3.2.2. *Teleportation from Example 3.1.2 can be defined as a function as follows:*

$$(tel') ::= \lambda * . (\mathbf{let} \langle y, q \rangle = \mathbf{Bell} \mathbf{in} \langle A(y), B(q) \rangle).$$

Now, we show some examples of type derivation for the examples of term originated from Example 3.1.2 and Example 3.1.2.

Example 3.2.3. Now, we show how the interpretation works on a typing derivation for the program exp in Eq. (1.1). We first show the typing derivation, then present the final result of the interpretation in the explicit form of tuple.

For the program exp , let's consider the following typing derivation:

$$\frac{\begin{array}{c} \vdots \\ \hline v_c : \mathbf{qubit} \vdash \text{meas}(v_c) : \mathbf{bool} \otimes \mathbf{qubit} \end{array} \quad \frac{\begin{array}{c} \vdots \\ \hline b : \mathbf{bool}, v_c : \mathbf{qubit} \vdash T : \mathbf{qubit} \otimes I \end{array}}{\hline v_c : \mathbf{qubit} \vdash exp : \mathbf{qubit} \otimes I}$$

where $T = \mathbf{if} \ b \ \mathbf{then} \ \langle \text{init}(tt), \text{free}(v_c) \rangle \ \mathbf{else} \ \langle v_c, * \rangle$.

$$\frac{\frac{\frac{\frac{\vdots}{\hline \vdash \text{unbox} : !A_m \multimap !(\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{qubit}))}}{\hline \vdash \text{meas} : !(\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{qubit}))}}{\hline \vdash \text{meas} : \mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{qubit}} \quad \frac{\vdots}{\hline v_c : \mathbf{qubit} \vdash v_c : \mathbf{qubit}}}{\hline v_c : \mathbf{qubit} \vdash \text{meas}(v_c) : \mathbf{bool} \otimes \mathbf{qubit}}$$

where

$$A_m = \text{QChan}(\mathbf{qubit}, \mathbf{bool} \otimes \mathbf{qubit})$$

$$\text{circ}_m = (x, \text{meas } x \in \{x\} \in \{x\}, [\langle tt, x \rangle, \langle ff, x \rangle]).$$

$$\frac{\frac{\frac{\frac{\frac{\vdash tt : \mathbf{bool}}{\hline x : \mathbf{qubit} \vdash \langle tt, x \rangle : \mathbf{bool} \otimes \mathbf{qubit}}}{\hline \mathbf{vBind}(\emptyset, \{x\}, \langle tt, x \rangle, \mathbf{bool} \otimes \mathbf{qubit})}}{\hline \mathbf{vBind}(\emptyset, \{x\}, \{x\}, [\langle tt, x \rangle, \langle ff, x \rangle], \mathbf{bool} \otimes \mathbf{qubit})}} \quad \frac{\frac{\frac{\frac{\vdash ff : \mathbf{bool}}{\hline x : \mathbf{qubit} \vdash \langle ff, x \rangle : \mathbf{bool} \otimes \mathbf{qubit}}}{\hline \mathbf{vBind}(\emptyset, \{x\}, \langle ff, x \rangle, \mathbf{bool} \otimes \mathbf{qubit})}}{\hline \mathbf{vBind}(\emptyset, \{x\}, \{x\}, [\langle tt, x \rangle, \langle ff, x \rangle], \mathbf{bool} \otimes \mathbf{qubit})}}}{\hline \mathbf{vBind}(\emptyset, \{x\}, \{x\}, [\langle tt, x \rangle, \langle ff, x \rangle], \mathbf{bool} \otimes \mathbf{qubit})}} \quad \frac{\emptyset = \emptyset}{\hline x \vDash \mathbf{qubit}}}{\hline \vdash (x, \text{meas } x \in \{x\} \in \{x\}, [\langle tt, x \rangle, \langle ff, x \rangle]) : !A_m}$$

$$\frac{\frac{\frac{\frac{\vdots}{\hline \vdash \text{init}(tt) : \mathbf{qubit}}}{\hline v_c : \mathbf{qubit} \vdash \langle \text{init}(tt), \text{free}(v_c) \rangle : \mathbf{qubit} \otimes I} \quad \frac{\frac{\frac{\vdots}{\hline v_c : \mathbf{qubit} \vdash \text{free}(v_c) : I}}{\hline v_c : \mathbf{qubit} \vdash v_c : \mathbf{qubit}} \quad \frac{\vdash * : I}{\hline v_c : \mathbf{qubit} \vdash \langle v_c, * \rangle : \mathbf{qubit} \otimes I}}{\hline v_c : \mathbf{qubit} \vdash \langle \text{init}(tt), \text{free}(v_c) \rangle : \mathbf{qubit} \otimes I} \quad \frac{\frac{\frac{\vdots}{\hline v_c : \mathbf{qubit} \vdash v_c : \mathbf{qubit}}}{\hline v_c : \mathbf{qubit} \vdash \langle v_c, * \rangle : \mathbf{qubit} \otimes I}}{\hline v_c : \mathbf{qubit} \vdash \langle v_c, * \rangle : \mathbf{qubit} \otimes I}}}{\hline b : \mathbf{bool}, v_c : \mathbf{qubit} \vdash \mathbf{if} \ b \ \mathbf{then} \ \langle \text{init}(tt), \text{free}(v_c) \rangle \ \mathbf{else} \ \langle v_c, * \rangle : \mathbf{qubit} \otimes I}$$

$$\frac{\frac{\frac{\frac{\frac{\frac{\vdash \text{unbox} : !\text{QChan}(I, \mathbf{qubit}) \multimap !(I \multimap \mathbf{qubit})}{\hline \vdash \text{unbox}(*, \text{init } b \ x \in \{x\}, x) : !(I \multimap \mathbf{qubit})}}{\hline \vdash \text{unbox}(*, \text{init } b \ x \in \{x\}, x) : I \multimap \mathbf{qubit}}}{\hline \vdash \text{unbox}(*, \text{init } b \ x \in \{x\}, x) : I \multimap \mathbf{qubit}} \quad \frac{\frac{\frac{\frac{\frac{\frac{\emptyset = \emptyset}{\hline x : \mathbf{qubit} \vdash x : \mathbf{qubit}}}{\hline * \vDash I} \quad \frac{\vdash (*, \text{init } b \ x \in \{x\}, x) : !\text{QChan}(I, \mathbf{qubit})}{\hline * \vDash I}}{\hline * \vDash I}}{\hline * \vDash I}}}{\hline * \vDash I}}}{\hline \vdash \text{init}(tt) : \mathbf{qubit}}$$

where

$$\frac{\vdots}{\frac{\vdash Z : \mathbf{qubit} \multimap \mathbf{qubit} \quad q : \mathbf{qubit} \vdash q : \mathbf{qubit}}{q : \mathbf{qubit} \vdash Z(q) : \mathbf{qubit}}}$$

$$\frac{\frac{x : \mathbf{bool} \vdash x : \mathbf{bool} \quad \vdots \quad y : \mathbf{bool}, q : \mathbf{qubit} \vdash CTRL_1 : \mathbf{qubit} \quad y : \mathbf{bool}, q : \mathbf{qubit} \vdash CTRL_2 : \mathbf{qubit}}{x : \mathbf{qubit}, y : \mathbf{qubit}, q : \mathbf{qubit} \vdash U_{XY} : \mathbf{qubit}}}{\text{where } CTRL_1 = \mathbf{if } y \mathbf{ then } X(Z(q)) \mathbf{ else } Z(q) \text{ and } CTRL_2 = \mathbf{if } y \mathbf{ then } X(q) \mathbf{ else } q.}$$

$$\frac{\frac{\frac{\frac{\vdots}{xy : \mathbf{bool} \otimes \mathbf{bool} \vdash xy : \mathbf{bool} \otimes \mathbf{bool}}{xy : \mathbf{bool} \otimes \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{let } \langle x, y \rangle = xy \mathbf{ in } U_{XY} : \mathbf{qubit}}{q : \mathbf{qubit} \vdash \lambda xy. (\mathbf{let } \langle x, y \rangle = xy \mathbf{ in } U_{XY}) : \mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit}}}{\vdash B : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})}}{\vdots}$$

$$\frac{\frac{\frac{\vdots}{y : \mathbf{qubit}, x : \mathbf{qubit} \vdash CNOT \langle x, y \rangle : \mathbf{QQ}}{y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \mathbf{let } \langle x, y \rangle = CNOT \langle x, y \rangle \mathbf{ in } \langle meas_f(H(x)), meas_f(y) \rangle : \mathbf{bool} \otimes \mathbf{bool}}}{y : \mathbf{qubit} \vdash \lambda x. (\mathbf{let } \langle x, y \rangle = CNOT \langle x, y \rangle \mathbf{ in } \langle meas_f(H(x)), meas_f(y) \rangle) : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})}}{\vdash A : \mathbf{qubit} \multimap (\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool})}}$$

$$\frac{\frac{\frac{\vdots}{\vdash meas_f : \mathbf{qubit} \multimap \mathbf{bool}}{x : \mathbf{qubit} \vdash meas_f(H(x)) : \mathbf{bool}} \quad \frac{\frac{\frac{\vdots}{\vdash H : \mathbf{qubit} \multimap \mathbf{qubit}}{x : \mathbf{qubit} \vdash H(x) : \mathbf{qubit}} \quad \frac{\vdots}{\vdash meas_f : \mathbf{qubit} \multimap \mathbf{bool}}}{y : \mathbf{qubit} \vdash meas_f(y) : \mathbf{bool}}}{y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \langle meas_f(H(x)), meas_f(y) \rangle : \mathbf{bool} \otimes \mathbf{bool}}}{\vdots}$$

$$\frac{\frac{\frac{\vdots}{\vdash CNOT : \mathbf{QQ} \multimap \mathbf{QQ}}{\vdash H : \mathbf{qubit} \multimap \mathbf{qubit}} \quad \frac{\frac{\vdots}{\vdash init(tt) : \mathbf{qubit}}{\vdash H(init(tt)) : \mathbf{qubit}} \quad \frac{\vdots}{\vdash init(tt) : \mathbf{qubit}}}{\vdash \langle H(init(tt)), init(tt) \rangle : \mathbf{qubit} \otimes \mathbf{qubit}}}{\vdash Bell : \mathbf{qubit} \otimes \mathbf{qubit}}$$

$$\frac{\frac{\vdots}{y : \mathbf{qubit} \vdash A(y) : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})} \quad \frac{\vdots}{q : \mathbf{qubit} \vdash B(q) : (\mathbf{bool} \otimes \mathbf{bool}) \multimap \mathbf{qubit}}}{\vdash \text{Bell} : \mathbf{QQ} \quad y : \mathbf{qubit}, q : \mathbf{qubit} \vdash \langle A(y), B(q) \rangle : (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes ((\mathbf{bool} \otimes \mathbf{bool}) \multimap \mathbf{qubit})}
\vdash \text{tel} : (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})$$

And for the variation of teleportation term of Example 3.2.2.

$$\frac{\vdots}{\vdash \text{tel} : (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})}
\vdash \text{tel}' : I \multimap (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})$$

In addition, the type system has the following properties. First of all, Lemma 3.2.3 states that: first, for a valid type judgement $!\Delta, Q \vdash M : A$, all free variables of the term M appears in the context $!\Delta, Q$ and linear variable in Q appears as free variable in M ; and, second, if a quantum channel constant (p, Q, m) is well binded, i.e. $\mathbf{vBind}(!\Delta, \text{out}(Q), m, A)$ is derivable, then the free variable of the quantum channel constant appears in the exponential context $!\Delta$.

Lemma 3.2.3. *The following two statements hold:*

- *If $(!\Delta, Q \vdash M : A)$, then $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q)))$.*
- *If $\mathbf{vBind}(!\Delta, \text{out}(Q), m, A)$, then $(\mathbf{FV}((p, Q, m)) \subseteq \mathbf{FV}(!\Delta))$*

As a consequence, $(\mathbf{FV}(Q) = \mathbf{FV}(M) \setminus \mathbf{FV}(!\Delta))$.

Proof. We prove these two properties by induction on the level of QChan.

Base case: When the level of QChan of m is -1 , both properties are true since there is no such m with level of QChan -1 .

Induction step: Assume that the properties hold for all term m with level of QChan i less than or equal to i . We show the two properties for any term m with QChan level $i + 1$ as follows.

1. If $(!\Delta, Q \vdash M : A)$, then $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q)))$.

Induction on type derivation.

- $(!\Delta, (x : A) \vdash x : A)$

We have that either $(Q = (x : A))$ or $(Q = \emptyset)$ and $(M = x)$, hence $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) = \{x\})$. Therefore, it follows that $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q)))$.

- $(! \Delta \vdash V : !A)$ when $(! \Delta \vdash V : A)$ and V is a value.
The goal directly follows from induction hypothesis.
- $(! \Delta, Q \vdash M : A)$ when $(! \Delta, Q \vdash M : !A)$
The goal directly follows from induction hypothesis.
- $(! \Delta \vdash * : I)$, $(! \Delta \vdash \text{tt} : \text{bool})$, or $(! \Delta \vdash \text{ff} : \text{bool})$ or the rules for (box) and (unbox)
We have that $(Q = \emptyset)$ and $(M = * (\text{tt}, \text{ or } \text{ff}))$, hence $(\mathbf{FV}(Q) = \mathbf{FV}(M) = \emptyset)$. Therefore, it follows that $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q)))$.
- $(! \Delta, Q_a, Q_b \vdash \text{if } M \text{ then } M_1 \text{ else } M_2 : A)$ where $(! \Delta, Q_a \vdash M : \text{bool})$, $(! \Delta, Q_b \vdash M_1 : A)$, and $(! \Delta, Q_b \vdash M_2 : A)$
From induction hypotheses, we obtain that $(\mathbf{FV}(Q_a) \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a)))$, $(\mathbf{FV}(Q_b) \subseteq \mathbf{FV}(M_1) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b)))$, and $(\mathbf{FV}(Q_b) \subseteq \mathbf{FV}(M_2) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b)))$.
Therefore, we can derive that $(\mathbf{FV}(Q_a, Q_b) = \mathbf{FV}(Q_a) \cup \mathbf{FV}(Q_b) \subseteq \mathbf{FV}(M) \cup \mathbf{FV}(M_1) \cup \mathbf{FV}(M_2) = \mathbf{FV}(\text{if } M \text{ then } M_1 \text{ else } M_2) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a) \cup \mathbf{FV}(Q_b)) = (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a, Q_b)))$.
- $(! \Delta, Q \vdash \lambda x.M : A_a \multimap A_b)$ where $(! \Delta, Q, (x : A_a) \vdash M : A_b)$
There are two cases to prove depending on A_a . First, let us assume that A_a is linear. Then, from induction hypothesis, we have that $(\mathbf{FV}(Q) \cup \{x\} \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q) \cup \{x\}))$. Therefore, it follows that $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) \setminus \{x\} = \mathbf{FV}(\lambda x.M) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q)))$ since $(x \notin \mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q))$.
Next, let us assume that A_a is non-linear. Then, induction hypothesis gives that $(\mathbf{FV}(Q) \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q) \cup \{x\}))$. Moreover, we have that $(x \notin \mathbf{FV}(Q))$. Therefore, we can derive that $(\mathbf{FV}(Q) = (\mathbf{FV}(Q) \setminus \{x\} \subseteq \mathbf{FV}(M) \setminus \{x\} = \mathbf{FV}(\lambda x.M) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q)))$ since $(x \notin \mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q))$.
- $(! \Delta, Q_a, Q_b \vdash \text{let } \langle x, y \rangle = M_a \text{ in } M_b : A)$ where $(! \Delta, Q_a \vdash M_a : A_1 \otimes A_2)$ and $(! \Delta, Q_b, (x : A_1), (y : A_2) \vdash M_b : A)$
From induction hypothesis, we can obtain that $(\mathbf{FV}(Q_a) \subseteq \mathbf{FV}(M_a) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a)))$ and $(\mathbf{FV}(Q_b) \cup V \subseteq \mathbf{FV}(M_b) \cup (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b) \cup V))$ where $(V \subseteq \{x, y\})$ depending on the linearity of x and y . It also follows that $(\mathbf{FV}(Q_b) \subseteq (\mathbf{FV}(M_b) \setminus \{x, y\}) \cup (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b)))$ since $(\mathbf{FV}(Q_b) \cap \{x, y\} = \emptyset)$ and $(\mathbf{FV}(! \Delta) \cap \{x, y\} = \emptyset)$.
Therefore, we can derive that $(\mathbf{FV}(Q_a, Q_b) = \mathbf{FV}(Q_a) \cup \mathbf{FV}(Q_b) \subseteq \mathbf{FV}(M_a) \cup (\mathbf{FV}(M_b) \setminus \{x, y\}) = \mathbf{FV}(\text{let } \langle x, y \rangle = M_a \text{ in } M_b) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a) \cup \mathbf{FV}(Q_b)) = (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a, Q_b)))$.
- $(! \Delta, Q_a, Q_b \vdash M_a M_b : A_b)$ where $(! \Delta, Q_a \vdash M_a : A_a \multimap A_b)$ and $(! \Delta, Q_b \vdash M_b : A_a)$

From induction hypotheses, we have that $(\mathbf{FV}(Q_a) \subseteq \mathbf{FV}(M_a) \subseteq (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_a)))$ and that $(\mathbf{FV}(Q_b) \subseteq \mathbf{FV}(M_b) \subseteq (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_b)))$.

Therefore, we can derive that $(\mathbf{FV}(Q_a, Q_b) = \mathbf{FV}(Q_a) \cup \mathbf{FV}(Q_b) \subseteq \mathbf{FV}(M_a) \cup \mathbf{FV}(M_b) = \mathbf{FV}(M_a M_b) \subseteq (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_a) \cup \mathbf{FV}(Q_b)) = (\mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_a, Q_b)))$.

- $(!\Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b)$ where $(!\Delta, Q_a \vdash M_a : A_a)$ and $(!\Delta, Q_b \vdash M_b : A_b)$

The same as the previous case.

- $(!\Delta \vdash (p, Q, m) : !\text{QChan}(P, A))$ where $(p \vDash P)$, and $(\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A))$

Since the QChan level of m is strictly less than the QChan level of (p, Q, m) , we can apply the induction hypothesis, which says that $(\mathbf{FV}((p, Q, m)) \subseteq \mathbf{FV}(!\Delta))$ if $(\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A))$.

Therefore, from the hypothesis, $(\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A))$, we can conclude that $(\mathbf{FV}((p, Q, m)) \subseteq \mathbf{FV}(!\Delta))$.

- $(\gamma_a \times \gamma_b \vdash [m_a, m_b] : A)$ where $(\gamma_a \vdash m_a : A)$ and $(\gamma_b \vdash m_b : A)$

Since the lemma concerns with the classical term, this case is irrelevant.

2. If $(\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A))$, then $(\mathbf{FV}((p, Q, m)) \subseteq \mathbf{FV}(!\Delta))$

We prove it by induction on Q .

- $(Q = \epsilon(V))$

First, we have that $(\mathbf{out}(Q) = V)$ and $m = M$.

Therefore, we know that the condition $(\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A))$ must be derived by using \mathbf{vBind}_{nb} , which implies that $(\mathbf{out}(Q) \cap \mathbf{FV}(!\Delta) = \emptyset)$ and $(!\Delta, \mathbf{TC}_Q(V) \vdash M : A)$.

Then applying the hypothesis, which says that if $(!\Delta, \mathbf{TC}_Q(V) \vdash M : A)$ then $(V \subseteq \mathbf{FV}(M) \subseteq (\mathbf{FV}(!\Delta) \cup V))$, we can obtain that $((\mathbf{FV}(M) \setminus V) \subseteq ((\mathbf{FV}(!\Delta) \cup V) \setminus V) = \mathbf{FV}(!\Delta))$.

- $(Q = U(V) Q', \text{init } b \text{ v } Q', \text{ or free } v \text{ } Q')$

In this case, we know that $(\mathbf{out}(Q) = \mathbf{out}(Q'))$, hence

$$(\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A) = \mathbf{vBind}(!\Delta, \mathbf{out}(Q'), m, A)).$$

However, we can derive that $(\mathbf{FV}((p, Q', m)) = \cup_i (\mathbf{FV}(m_i) \setminus \mathbf{out}(Q')_i) = \cup_i (\mathbf{FV}(m_i) \setminus \mathbf{out}(Q)_i) = \mathbf{FV}((p, Q, m)))$.

Then, by the induction hypothesis, we have that $(\mathbf{FV}((p, Q', m)) \subseteq \mathbf{FV}(!\Delta))$.

Therefore, we can conclude that $(\mathbf{FV}((p, Q, m)) \subseteq \mathbf{FV}(!\Delta))$.

- $Q = \text{meas } v \ Q_1 \ Q_2$

We know that the hypothesis ($\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m, A)$) is derived by the rule \mathbf{vBind}_b , which implies that

- $m = [m_a, m_b]$,
- $(\mathbf{out}(Q) = [\mathbf{out}(Q_1), \mathbf{out}(Q_2)])$,
- $(\mathbf{vBind}(!\Delta, c_a, m_a, A))$, and
- $(\mathbf{vBind}(!\Delta, c_b, m_b, A))$.

By induction hypothesis, we obtain that $(\mathbf{FV}((p, Q_1, m_a)) \subseteq \mathbf{FV}(!\Delta))$ and $(\mathbf{FV}((p, Q_2, m_b)) \subseteq \mathbf{FV}(!\Delta))$.

Therefore, since $(\mathbf{FV}((p, Q, m)) = \mathbf{FV}((p, Q_1, m_a)) \cup \mathbf{FV}((p, Q_2, m_b))) \subseteq \mathbf{FV}(!\Delta)$.

□

Secondly, Lemma 3.2.4 states that if a value has an exponential type, then it does not contain any linear free variables.

Lemma 3.2.4. *Suppose that $(!\Delta, Q \vdash V : !A)$. Then Q is empty.*

Proof. We proceed by induction on the derivation of $!\Delta, Q \vdash V : !A$. The relevant cases are:

- If V is a variable, it is trivially true.
- If $(!\Delta, Q \vdash V : !A)$ is derived from (d): then by the induction hypothesis, Q is empty.
- Cases of (p) and (QChan) are trivial

All other cases are irrelevant: We, therefore, have Q empty. □

Next, the substitution lemma (Lemma 3.2.5) states that the type of a term is preserved when we substitute a variable in the typed term with another term with the type of the variable.

Lemma 3.2.5. *Consider a branching term m . Then the two following properties hold:*

- *If $m = M$, $(!\Delta, Q_a, (x : A_a) \vdash M : A_b)$ and $(!\Delta, Q_b \vdash V : A_a)$, then $(!\Delta, Q_a, Q_b \vdash M[V/x] : A_b)$.*
- *If $\mathbf{vBind}(!\Delta, x : !A', c, b, m, A)$ and $!\Delta \vdash V : !A'$ is a value then $\mathbf{vBind}(!\Delta, c, b, m[V/x], A)$.*

Proof. Induction on the QChan level of m .

Base case: QChan level is -1 : trivial since there is no such m .

Induction case: Suppose that it is true for all m with QChan level $n - 1$, and pick m with QChan level n .

1. Let us prove the first bullet: assume that $m = M$ is non-branching, that $(! \Delta, Q_a, (x : A_a) \vdash M : A_b)$ and that $(! \Delta, Q_b \vdash V : A_a)$. We want to show that $(! \Delta, Q_a, Q_b \vdash M[V/x] : A_b)$.

Induction on typing derivation of $(! \Delta, Q_a, (x : A_a) \vdash M : A_b)$.

- $(! \Delta, (x : A) \vdash x : A)$

First, it can be derived that: either $(M = x)$, $(Q_a = \emptyset)$, and $(A_a = A_b)$; or $(M = y)$, $Q_a = (y : A_b)$, and A_a is non-linear type.

For the first case, since $(M = x)$, we obtain that $(M[V/x] = V)$. Therefore, the hypothesis, $(! \Delta, Q_b \vdash V : A_a)$, implies that $(! \Delta, Q_a, Q_b \vdash M[V/x] : A_b)$.

For the second case, since A_a is linear, from Lemma 3.2.4, we know that $(Q_b = \emptyset)$. Therefore, $! \Delta, (y : A_b) \vdash y : A_b$ can be obtained by type rule (var), and, hence, $! \Delta, Q_a, Q_b \vdash M[V/x] : A_b$ as well.

- $(! \Delta', Q \vdash M : A)$ where $(! \Delta', Q \vdash M : !A)$

First, it can be derived that $((! \Delta, Q_a, (x : A_a)) = (! \Delta', Q))$ and $(A_b = A)$.

By induction hypothesis, we obtain that $(! \Delta, Q_a, Q_b \vdash M[V/x] : !A)$.

Then, by applying the typing rule, we can conclude that $(! \Delta, Q_a, Q_b \vdash M[V/x] : A)$.

- $(! \Delta' \vdash V' : !A)$ where $(! \Delta' \vdash V' : A)$ and V' is a value

First, we have that $(! \Delta' = ! \Delta, Q_a, (x : A_a))$, $(M = V')$, and $(!A = A_b)$.

We can deduce that Q_a is empty and that A_a is of the form $!A'_a$.

Therefore we have $! \Delta, Q_b \vdash V : !A'_a$. From Lemma 3.2.4 we deduce that Q_b is empty.

Then, by induction hypothesis, we obtain that $(! \Delta \vdash V'[V/x] : A)$.

From Lemma 3.1.6, the term $V'[V/x]$ is a value: we can apply (p) and get $(! \Delta \vdash V'[V/x] : !A)$.

- Case (I): the term M is $*$, Q_a is empty, and A_a is of the form $!A'_a$. From Lemma 3.2.4, We conclude that Q_b is empty. Therefore, since $*[V/x] = *$, the result is true.

- For all the other constant cases: tt, ff, box, unbox, the argument is similar.
- (p, Q, m) : the derivation starts with (QChan_I) , and we have Q_a empty, A_a of the form $!A'_a$, A_b of the form $!\text{QChan}(P, A)$ and $p \vDash P$. We also have that

$$\mathbf{vBind}(!\Delta, x : !A'_a, \mathbf{out}(Q), m, A)$$

From Lemma 3.2.4, Q_b is empty. We therefore want to show that

$$!\Delta \vdash (p, Q, m)[V/x] : !\text{QChan}(P, A).$$

We have $(p, Q, m)[V/x] = (p, Q, m[V/x])$.

We already know that $p \vDash P$. To be able to use (QChan_I) we only need to make sure that

$$\mathbf{vBind}(!\Delta, \mathbf{out}(Q), m[V/x], A).$$

Since m has a QChan level strictly smaller than the one of M , we can invoke the induction hypothesis to conclude.

- (\multimap_I) . The term M is $\lambda y.M'$ (where we can wlog assume that $x \neq y$ and that $y \notin |Q_b|$), A_b is the form $A \multimap A'$, and $!\Delta, Q_a, x : A_a, y : A \vdash M' : A'$. By induction hypothesis $!\Delta, Q_a, Q_b, y : A \vdash M'[V/x] : A'$, and we can conclude with rule (\multimap) .
- (\multimap_E) . The term M is of the form $M_1 M_2$ and $Q_a = Q_1, Q_2$. There are three cases for $x : A_a$:
 - Either A_a is of the form $!A'_a$. Then $!\Delta, x : !A'_a, Q_1 \vdash M_1 : A \multimap A_b$ and $!\Delta, x : !A'_a, Q_2 \vdash M_2 : A$ for some type A . One can apply the induction hypothesis on both typing judgements and get $!\Delta, Q_1 \vdash M_1[V/x] : A \multimap A_b$ and $!\Delta, Q_2 \vdash M_2[V/x] : A$. Applying Rule (\multimap_E) gives the required result.
 - Or A_a is linear and x only appear in M_1 . Then $!\Delta, x : !A'_a, Q_1 \vdash M_1 : A \multimap A_b$ and $!\Delta, Q_2 \vdash M_2 : A$. One can apply the induction hypothesis on both typing judgements and get $!\Delta, Q_1 \vdash M_1 : A \multimap A_b$ and $!\Delta, Q_2 \vdash M_2[V/x] : A$. Applying Rule (\multimap_E) gives the required result.
 - Or A_a is linear and x only appear in M_2 . This is similar to the previous case.
- The cases $\langle M_a, M_b \rangle$, **let** $\langle x, y \rangle = M_a$ **in** M_b and **if** M **then** M_a **else** M_b are completely similar.

2. Now we prove the second bullet of the lemma: Assume that $\mathbf{vBind}(!\Delta, x : !A', c, m, A)$ and $!\Delta \vdash V : !A'$ is a value then $\mathbf{vBind}(!\Delta, c, m[V/x], a)$. We proceed by structural induction on m .

- Either $m = M$ is non-branching. Then $\mathbf{vBind}(!\Delta, x : !A', c, m, A)$ is derived from Rule (\mathbf{vBind}_{nb}): we know that
 - $c \cap \mathbf{FV}(!\Delta, x : !A') = \emptyset$
 - $!\Delta, x : !A', \mathbf{TC}_Q(c) \vdash M : A$

We can derive that $\mathbf{vBind}(!\Delta, c, m[V/x], a)$ provided that (using again Rule (\mathbf{vBind}_{nb})):

- $c \cap \mathbf{FV}(!\Delta) = \emptyset$: we already know since $c \cap \mathbf{FV}(!\Delta, x : !A') = \emptyset$
- $!\Delta, \mathbf{TC}_Q(c) \vdash M[V/x] : A$.

From the induction hypothesis applied on $(!\Delta, x : !A', \mathbf{TC}_Q(c) \vdash M : A)$ and $(!\Delta \vdash V : !A')$, we derive that $(!\Delta, \mathbf{TC}_Q(c) \vdash M[V/x] : A)$.

- Or $m = [m_a, m_b]$. Then $c = [c_a, c_b]$. The property

$$\mathbf{vBind}(!\Delta, x : !A', [c_a, c_b], [m_a, m_b], A)$$

is derived from Rule (\mathbf{vBind}_b): we know that

$$\mathbf{vBind}(!\Delta, x : !A', c_a, m_a, A), \quad \mathbf{vBind}(!\Delta, x : !A', c_b, m_b, A).$$

We can apply induction hypothesis on both formulas and then conclude with Rule (\mathbf{vBind}_b) since $m[V/x] = [m_a[V/x], m_b[V/x]]$.

This closes the proof of the substitution lemma. □

3.3 . Operational Semantics

The operational semantics of the language is formalized by the abstract machine with the transition function over the states, called configuration. The functional behaviors of the operators and constructors of the term are encoded in the reduction rules. The computational model we have in mind for the language is the QRAM model.

In the quantum lambda-calculus, the corresponding operational semantics is designed with the use of an abstract machine of the form (ϕ, L, M) where ϕ is the state of the QRAM, M is the term under consideration, and L is a function binding the free variables of M to the qubits in ϕ . In the case of Proto-Quipper, another abstract machine is considered. Indeed, the computational model is not based on the QRAM but specialized in circuit construction: the operational semantics is modeling a form of I/O side-effect: gates are emitted and buffered in a circuit.

In our language, we have both access to a QRAM and the capability to manipulate generalized circuits in the form of quantum channels. We then define two operational semantics to capture the two possible behavior of the language: one semantics is designed for buffering and manipulating quantum channels, while the other one is for running the quantum channel in the global environment.

3.3.1 . Circuit-buffering operational semantics

Configuration

Following the formalization of the operational semantics in Proto-Quipper [56], the circuit-buffering abstract machine operates on the quantum channel while reducing the term. Naturally, the configuration is represented by a pair (Q, m) consisting of a quantum channel Q and a branching term m . The two branching structures are supposed to match. Note that we do not need a binding function that maps each wire in the output of Q to the free variables in m since we identify the space of wire names for quantum channel objects and variable names for the term.

However, since we introduce the classical context, the configuration of the circuit-buffering operational semantics can have free classical variables. Hence, the free variables of term M are not necessarily connected to the output wires of the buffered circuit. On the other hand, all the wires from the output of the circuit need to appear once in the term. These conditions of configuration is formalized in Definition 3.3.1. For convenience, let us divide the free variables of the term in a circuit-buffering configuration into two parts: quantum variables, which are the output wires of the quantum channel object, and classical variables, which are not quantum variables.

This definition of valid circuit-buffering configuration is similar to the definition of valid quantum channel constant. One can notice that a circuit-buffering configuration corresponds to a quantum channel constant without the input wires. Actually, each valid circuit-buffering configuration can be obtained from valid quantum channel constants of different patterns of input wires, as stated in Lemma 3.3.1.

Definition 3.3.1 (Notion of valid circuit-buffering configuration). A circuit-buffering configuration is a pair (Q, m) . Let us define a binary relation \mathbf{vc} on Q and m stating that they share the same tree-structure:

$$\frac{V \subseteq \mathbf{FV}(M)}{\mathbf{vc}(\epsilon(V), M)} \quad \frac{\mathbf{valid}(Q') \quad \mathbf{vc}(Q, m)}{\mathbf{vc}(Q', m)}$$

$$\frac{\mathbf{valid}(\text{meas } w \ Q_1 \ Q_2) \quad \mathbf{vc}(Q_1, m_a) \quad \mathbf{vc}(Q_2, m_b)}{\mathbf{vc}(\text{meas } w \ Q_1 \ Q_2, [m_a, m_b])}$$

where Q' refers to $U(V) \ Q$, $\text{init } b \ v \ Q$, and $\text{free } v \ Q$. We ignore the name of the relation \mathbf{vc} for valid circuit-buffering configuration unless it is necessary.

Lemma 3.3.1. *Provided (p, Q, m) is a quantum channel constant, (Q, m) is valid.*

Graphical representation of the circuit-buffering configuration We use a graphical representation for the circuit-buffering configuration. A green box represents a quantum channel whose leaves are linked to square-boxed terms. The edges represent bundles of wires, which can contain multiple wires and be empty. For instance, the circuit-buffering configuration $(\text{meas } q \in(\{q\}) \in(\{q\}), [\langle \text{tt}, q \rangle, \langle \text{ff}, q \rangle])$ is represented with the diagram in Figure 3.1.

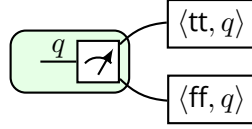


Figure 3.1: Graphical representation of the circuit-buffering configuration $(\text{meas } q \in(\{q\}) \in(\{q\}), [\langle \text{tt}, q \rangle, \langle \text{ff}, q \rangle])$

Equivalence over configurations We define the equality of circuit-buffering configuration based on the renaming of closed variables of the term (note that the term is already defined as the equivalence classes over the α -equivalence) and the renaming of wire names in the quantum channel object. The equivalence requires that the quantum channel objects and the terms in two configurations be equivalent over the same renaming function.

In order to define the equality of the configuration, we first define an auxiliary equivalence relation depending on a binding function (a bijection) for input wires. As defined in Definition 3.3.2, the auxiliary equivalence means that two circuit-buffering configurations are equivalent over the renaming operation. In the definition, we use the renaming operator $\text{rename}(m, V, f)$ applied to a term m given a renaming function f and a set of variable V as the term obtained by renaming of variables in V in m by the renaming function f restricted to the set of variable V .

Definition 3.3.2 (Equivalence of the circuit-buffering configuration over renaming operation over f, \sim_f). In order to define the equivalence relation of the circuit-buffering configuration, we first define the auxiliary equivalence relation \sim_f of it, which depends on wire name renaming, which is defined inductively as follows.

$$\frac{\epsilon(V_a) \sim_f \epsilon(V_b) \quad M_a \sim \mathbf{rename}(M_b, V_b, f) \quad M_b \sim \mathbf{rename}(M_a, V_a, f^{-1})}{((\epsilon(V_a), M_a) \sim_f (\epsilon(V_b), M_b))}$$

where f is a bijection from V_b to V_a and $\mathbf{rename}(m, V, f)$ means the term

obtained by renaming the variables V in m by f .

$$\frac{(Q'_1, m_a) \sim_f (Q'_2, m_b)}{((U(f(V)) Q'_1, m_a) \sim_f (U(V) Q'_2, m_b))} \quad \frac{(Q'_1, m_a) \sim_{f \cup \{v_2 \mapsto v_1\}} (Q'_2, m_b)}{(\text{init } b \ v_1 \ Q'_1, m_a) \sim_f (\text{init } b \ v_2 \ Q'_2, m_b)}$$

$$\frac{(Q'_1, m_a) \sim_{f \setminus \{v \mapsto f(v)\}} (Q'_2, m_b)}{(\text{free } f(v) \ Q_1, m_a) \sim_f (\text{free } v \ Q_2, m_b)}$$

$$\frac{(Q_1, m_a) \sim_f (Q_3, m_c) \quad (Q_2, m_b) \sim_f (Q_4, m_d)}{(\text{meas } f(v) \ Q_1 \ Q_2, [m_a, m_b]) \sim_f (\text{meas } v \ Q_3 \ Q_4, [m_c, m_d])}$$

Then, we define the equivalence of circuit-buffering configuration \sim as the subset of the auxiliary equivalence relation where the renaming function is identity function \sim_{id} .

We can check that if $(Q_1, m_a) \sim_f (Q_2, m_b)$, then $\text{in}(Q_1) = f(\text{in}(Q_2))$. The subscript f is ignored when it is an identity function which is the case for the equivalence relation. Therefore, it follows that if $(Q_1, m_a) \sim (Q_2, m_b)$, then $\text{in}(Q_1) = \text{in}(Q_2)$ since they are equivalent over the identity. Moreover, it can be checked that the equivalence relation of configuration over the identity function is reflexive, symmetric, and transitive.

Although the equivalence relation of circuit-buffering configuration implies that input wires of the quantum channel objects Q_1 and Q_2 in the equivalent configuration are equal, they can have different output wires since the definition allows for renaming auxiliary qubits created inside the quantum channel objects. Also, note that we do not allow the renaming of free classical variables in the term in the auxiliary equivalence relation. Therefore, one can consider the equivalence relation is an extension of α -equivalence of the term to the circuit-buffering configuration where the auxiliary wires created in the quantum channel object are considered as closed variables.

Reduction rules

The reduction rules are defined as a transition function over the configuration of the abstract machine for circuit-buffering. We let the right arrow \rightarrow refer to the reduction over the configuration of the abstract machine. This relation, defined over the equivalence class of the circuit-buffering configurations by Definition 3.3.2, depicts the operational semantics of the execution of each term construction, which interacts with the quantum channel object in context.

The reduction rules are composed of different sets of rules (Definitions 3.3.4 – 3.3.8) which are called–reduction rules for classical computation and circuit operators; and structural reduction rules for quantum channel constants, empty quantum channel and non-empty quantum channel. The reduction rules for the terms with structural construction rules with an empty quantum

channel (Definition 3.3.7) require the extend operation on the quantum channel object (Definition 3.1.11) to represent the composition of a configuration and another configuration with a non-branching term. Moreover, the same set of rules requires for the composition of terms an extended notion of the term to represent the composition of a branching term and a non-branching term.

Definition 3.3.3 (Extending the term notation). Let us extend the notations for term constructs in the following way.

$$\begin{aligned}
M[m_a, m_b] &:= [Mm_a, Mm_b] \\
[m_a, m_b]M &:= [m_aM, m_bM] \\
\langle M, [m_a, m_b] \rangle &:= [\langle M, m_a \rangle, \langle M, m_b \rangle] \\
\langle [m_a, m_b], M \rangle &:= [\langle m_a, M \rangle, \langle m_b, M \rangle] \\
\mathbf{if} [m_a, m_b] \mathbf{then} M \mathbf{else} N &:= [\mathbf{if} m_a \mathbf{then} M \mathbf{else} N, \mathbf{if} m_b \mathbf{then} M \mathbf{else} N] \\
\mathbf{let} \langle x, y \rangle = [m_a, m_b] \mathbf{in} N &:= [\mathbf{let} \langle x, y \rangle = m_a \mathbf{in} N, \mathbf{let} \langle x, y \rangle = m_b \mathbf{in} N]
\end{aligned}$$

Next, the reduction rules are defined the same, except that the composed terms in structural reduction rules for the empty quantum channel can have common free variables (classical).

Rules (a.x) always hold (b ranges over $\{\text{tt}, \text{ff}\}$). In Rules (b.1), p is a pattern of same shape as P made from dynamically allocated fresh variables. In Rule (b.2), p and V have the same shape, and σ is a substitution mapping p to V . Provided that $(Q, m) \rightarrow (Q', m')$, we have $(\epsilon(\emptyset), (p, Q, m)) \rightarrow (\epsilon(\emptyset), (p, Q', m'))$. Provided that we have that $(\epsilon(W_M), M) \rightarrow (Q, m)$, that $\text{all}(Q) \cap W_N = \emptyset$ and that $\text{all}(Q) \cap W_V = \emptyset$, the class of rules (c) apply. There, $C[-]$ ranges over $[-]N$, $V[-]$, $\langle [-], N \rangle$, $\langle V, [0] \rangle$, $\mathbf{if} [-] \mathbf{then} M_a \mathbf{else} M_b$ and $\mathbf{let} \langle x, y \rangle = [-] \mathbf{in} N$. We use syntactic sugar for combining terms and branching terms, as in $C[m]$. It corresponds to the term constructor applied to each leaf of m , for instance: for $m = [[N_1, N_2], N_3]$, $C[m] := [[C[N_1], C[N_2]], C[N_3]]$. In Rules (d.x), Q stands for $\text{meas } w \ Q_1 \ Q_2$ and Q' for $\text{meas } w \ Q_3 \ Q_4$. These rules apply whenever $(Q_1, m_a) \rightarrow (Q_3, m_c)$ and $(Q_2, m_b) \rightarrow (Q_4, m_d)$. In (d.3), G ranges over $U(W)$, $\text{init } b \ w$ and free w .

Definition 3.3.4 (Reduction rules for classical computation). Reduction rules for classical computation show the reduction of the function application, let binding, and the conditional statements. Note that structural reduction rules for classical computation can be found in the paragraph for structural rules for empty quantum channel.

$$\begin{array}{c}
\hline
(\epsilon(W), (\lambda x.M)V) \rightarrow (\epsilon(W), M[V/x]) \\
\hline
\hline
(\epsilon(W), \mathbf{let} \langle x, y \rangle = \langle V, U \rangle \mathbf{in} M) \rightarrow (\epsilon(W), M[V/x, U/y]) \\
\hline
\hline
\end{array}$$

$$\frac{}{(\epsilon(W), \mathbf{if\ tt\ then\ } M_a \mathbf{\ else\ } M_b) \rightarrow (\epsilon(W), M_a)}$$

$$\frac{}{(\epsilon(W), \mathbf{if\ ff\ then\ } M_a \mathbf{\ else\ } M_b) \rightarrow (\epsilon(W), M_b)}$$

Definition 3.3.5 (Reduction rules for circuit operations). Reduction rules for circuit operations show the semantics of the application box and unbox operators. We let $\mathbf{new}(W, P)$ be a new pattern of type P such that the support has no common element with W and $\mathbf{bind}(p, V)$ be the binding function from $\text{supp}(p)$ to $\text{FV}(V)$ where V is a bundle of variables.

$$\frac{p = \mathbf{new}(P) \quad W_p = \mathbf{supp}(p)}{(\epsilon(\emptyset), \mathbf{box}_P V) \rightarrow (\epsilon(\emptyset), (p, \epsilon(W_p), Vp))}$$

where \mathbf{new} operator creates during the computation free variables, meaning that the free variables do not appear in both classical and quantum contexts. The term $\mathbf{new}(P)$ is a pattern of same shape as P , made out of these new variables.

$$\frac{\mathbf{shape}(p) = \mathbf{shape}(V) \quad \sigma = \mathbf{bind}(p, V)}{(\epsilon(\mathbf{FV}(V)), (\mathbf{unbox}(p, Q, u))V) \rightarrow (\sigma(Q), \sigma(u))}$$

Lemma 3.3.2. *Provided that P is a pattern-type and $p = \mathbf{new}(P)$, we have $\mathbf{TC}_Q(\mathbf{FV}(p)) \vdash p : P$* \square

Definition 3.3.6 (Structural reduction rule for quantum channel constant). Structural rules for quantum channel constant allows us to reduce the term inside a quantum channel constant.

$$\frac{(Q, m) \rightarrow (Q', m')}{(\epsilon(\emptyset), (p, Q, m)) \rightarrow (\epsilon(\emptyset), (p, Q', m'))}$$

Definition 3.3.7 (Structural reduction rules for empty quantum channel). Structural rules for empty quantum channel show the reduction of term which relies on the reduction of structural constructors of the language (i.e. function application, pair, let, and if constructors). Since the term can reduce to a branching term, the structural rules for empty quantum channel needs to be extended to branching terms which consists of terms sharing the same structure. The resulting branching term can be defined by using tcps defined above.

$$\frac{(\epsilon(W_M), M) \rightarrow (Q, m) \quad \mathbf{all}(Q) \cap W_N = \emptyset}{(\epsilon(W_M \cup W_N), MN) \rightarrow (\mathbf{extend}(Q, W_N), mN)}$$

$$\frac{(\epsilon(W_M), M) \rightarrow (Q, m) \quad \mathbf{all}(Q) \cap W_N = \emptyset}{(\epsilon(W \cup W_N), NM) \rightarrow (\mathbf{extend}(Q, W_N), Nm)}$$

$$\frac{(\epsilon(W_M), M) \rightarrow (Q, m) \quad \mathbf{all}(Q) \cap W_N = \emptyset}{(\epsilon(W_M \cup W_N), \langle M, N \rangle) \rightarrow (\mathbf{extend}(Q, W_N), \langle m, N \rangle)}$$

$$\frac{(\epsilon(W_M), M) \rightarrow (Q, m) \quad \mathbf{all}(Q) \cap W_N = \emptyset}{(\epsilon(W_M \cup W_N), \langle N, M \rangle) \rightarrow (\mathbf{extend}(Q, W_N), \langle N, m \rangle)}$$

$$\frac{(\epsilon(W_M), M) \rightarrow (Q, m) \quad \mathbf{all}(Q) \cap W_N = \emptyset}{(\epsilon(W_M \cup W_N), \mathbf{if } M \mathbf{ then } M_a \mathbf{ else } M_b) \rightarrow (\mathbf{extend}(Q, W_N), \mathbf{if } m \mathbf{ then } M_a \mathbf{ else } M_b)}$$

$$\frac{(\epsilon(W_M), M) \rightarrow (Q, m) \quad \mathbf{all}(Q) \cap W_N = \emptyset}{(\epsilon(W_M), \mathbf{let } \langle x, y \rangle = M \mathbf{ in } N) \rightarrow (\mathbf{extend}(Q, W_N), \mathbf{let } \langle x, y \rangle = m \mathbf{ in } N)}$$

Definition 3.3.8 (Structural reduction rules for non-empty quantum channel). Reduction can take place in each branch of a branching term. The structural rules for non-empty quantum channel let us find the reducible term within the branching term. There can be different reduction strategies regarding the measurement. The proposed rules show an example of strategy where all reducible terms of the branching term reduces in one step. This set of structural rules allows us to focus on reductions over empty quantum channels.

$$\frac{(Q_1, m_a) \rightarrow (Q_3, m_c) \quad (Q_2, m_b) \rightarrow (Q_4, m_d)}{((\mathbf{meas } w Q_1 Q_2), [m_a, m_b]) \rightarrow ((\mathbf{meas } w Q_3 Q_4), [m_c, m_d])}$$

$$\frac{(Q_1, m_a) \rightarrow (Q_3, m_c)}{((\mathbf{meas } w Q_1 Q_2), [m_a, v]) \rightarrow ((\mathbf{meas } w Q_3 Q_2), [m_c, v])}$$

$$\frac{(Q_2, m_b) \rightarrow (Q_4, m_d)}{((\mathbf{meas } w Q_1 Q_2), [v, m_b]) \rightarrow ((\mathbf{meas } w Q_1 Q_4), [v, m_d])}$$

$$\frac{(Q_1, m_a) \rightarrow (Q_2, m_b)}{(Q'_1, m_a) \rightarrow (Q'_2, m_b)}$$

where Q' is any one from $U(W) Q$, $\text{init } b w Q$, or $\text{free } w Q$.

As discussed before, proof normalization in linear logic is related to the reduction rules. In the proof normalization, type derivation is contracted by some rules like cut-elimination. In this context, there is no notion of the quantum channel object in the circuit-buffering configuration. However, in order to formalize the operational semantics of a program, we have introduced the quantum channel object to represent the context, i.e., the communication between the classical host and the quantum co-processor. The reduction rules are, therefore, obtained by adjusting the proof normalization rules and by adding some additional reduction rules for the quantum channel operators.

Regarding the reduction, we show the following lemma. Intuitively, the lemma states that free classical variables and dangling quantum variables are preserved over the equivalence of circuit-buffering configuration.

Examples

To illustrate the operational semantics and its reduction rules for circuit-buffering operational semantics, let us take the examples from the previous section—the non-trivial branching term with measurement (Example 3.1.1) and the quantum teleportation (Example 3.1.2).

Example 3.3.1. *Let us explain how the tree expands as the computation progresses for the term in Example 3.1.1. Figure 3.2 summarizes the reduction of the example. In the first line, the measurement in the term is reduced by the structural rule for let and the reduction rule for measurement creating a branching term. Then, each term at a leaf of the tree is reduced into the left-most configuration of the second line. Note how classical computation can happen inside the leaves. The second line of the figure shows the application of initialization and free operation. In particular, note how the tree expands as the computation progresses.*

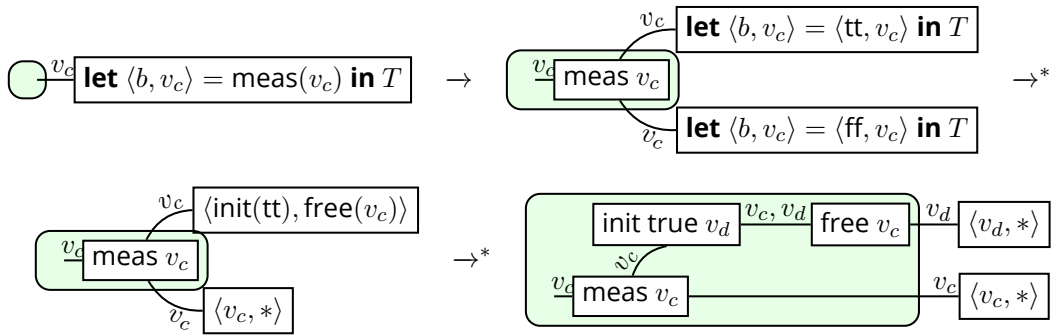
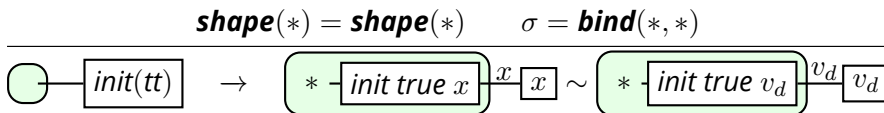


Figure 3.2: Reduction of the term of Example 3.1.1

Now, let us explain how we derive the reduction in the second line of Figure 3.2 from the reduction rules in Definitions 3.3.4 – 3.3.8.

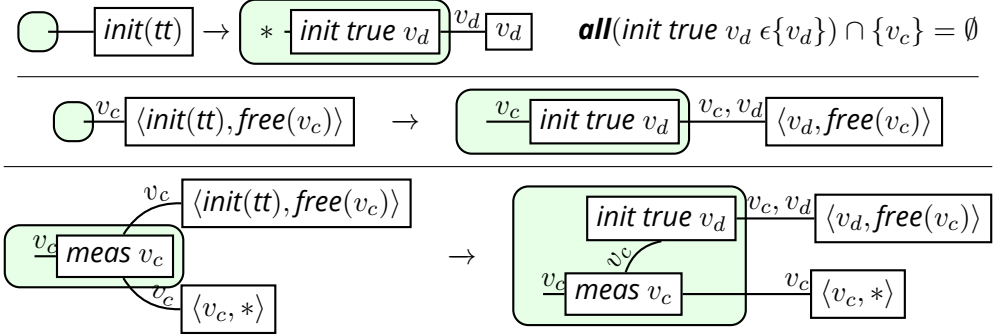
First, we show that $((\epsilon\{\}, \text{init}(\text{tt})) \rightarrow ((\text{init true } x \in\{x\}, x))$ as follows.



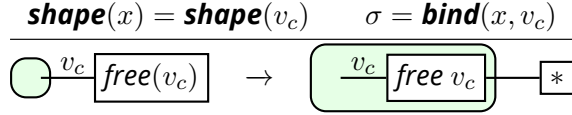
where we let

$$\text{init}(\text{tt}) = \text{unbox} \left(\ast, \ast \text{ -- } \boxed{\text{init true } x} \text{ -- } x, x \right) (\ast).$$

Then we can show the following reduction:



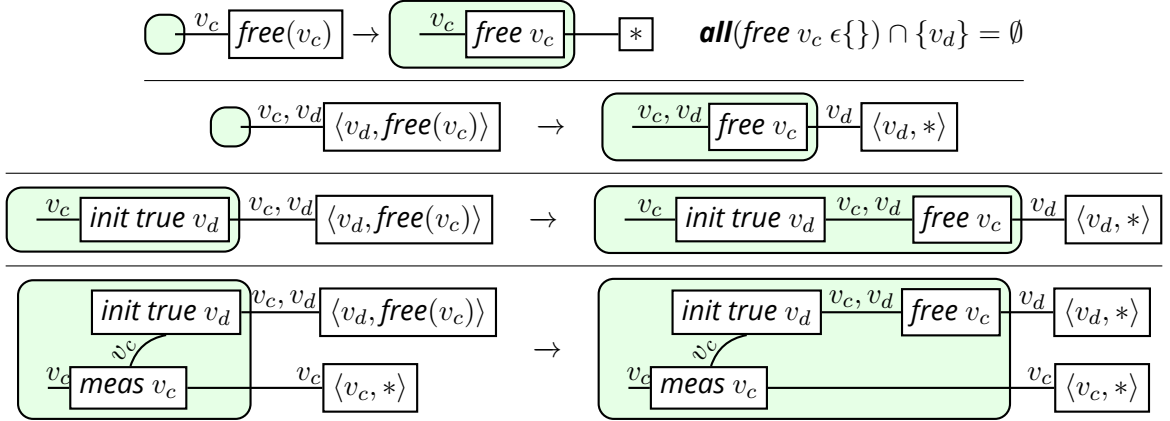
Next, we show the last reduction step of the example as follows.



Recall that

$$\mathbf{free} = \mathbf{unbox} \left(x, \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \text{---} \right).$$

Then we can show the following reduction:



3.3.2 . QRAM-based operational semantics

In the quantum simulation, the quantum co-processor simulates the physical system according to the quantum circuit generated by the classical host and returns the value to the host when it measures the state. To formalize the simulation, we define an abstract machine that maintains the current state of the quantum state according to the quantum channel object transmitted by the unbox operator. Each trace of the quantum channel object is combined with a non-branching term. The machine simulates and reduces the quantum channel object by using the algebraic structure, Q-coproc.

The state of the abstract machine

The state of the abstract machine consists of the quantum state and the buffered quantum channel object. The state of the buffer requires all information from the state of the circuit-buffering abstract machine, namely, the quantum channel object and the term.

Definition 3.3.9. The state of the quantum abstract machine is the triple of the quantum state, the quantum channel object, and the term represented as $[Q, C, M]$.

The reduction rules

Next, we define probabilistic reduction rules for the quantum simulation. The quantum co-processor simulates the quantum state consuming the quantum channel object. The reduction creates mixed states, which are the probability distributions of pure states. Specifically, the probabilities are given by the operator of the algebraic structure, Q-coproc.

Definition 3.3.10. The probabilistic reduction rules for the quantum abstract machine, \xRightarrow{p} , is defined as follows. The reduction $S_1 \xRightarrow{p} S_2$ can be read S_1 reduces to S_2 with probability p .

$$\frac{(U, w_1, C') = \text{pop}(C) \quad Q' = \text{Unitary}_n(U(w_1), w_1, Q)}{[Q, C, M] \xRightarrow{1} [Q', C', M]}$$

$$\frac{(\text{init}, b_0, w, C') = \text{pop}(C) \quad Q' = \text{Init}(b_0, w, Q)}{[Q, C, M] \xRightarrow{1} [Q', C', M]}$$

$$\frac{(\text{meas}, w, C_1, C_2) = \text{pop}(C) \quad (Q_1, p_1, Q_2, p_2) = \text{Meas}(Q, w) \quad M = [M_1, M_2]}{[Q, C, M] \xRightarrow{p_1} [Q_1, C_1, M_1]}$$

$$\frac{(\text{meas}, w, C_1, C_2) = \text{pop}(C) \quad (Q_1, p_1, Q_2, p_2) = \text{Meas}(Q, w) \quad M = [M_1, M_2]}{[Q, C, M] \xRightarrow{p_2} [Q_2, C_2, M_2]}$$

$$\frac{(\text{free}, w, C') = \text{pop}(C) \quad (Q_1, p_1, Q_2, p_2) = \text{Free}(w, Q)}{[Q, C, M] \xRightarrow{p_1} [Q_1, C', M]}$$

$$\frac{(\text{free}, w, C') = \text{pop}(C) \quad (Q_1, p_1, Q_2, p_2) = \text{Free}(w, Q)}{[Q, C, M] \xrightarrow{p_1} [Q_2, C', M]}$$

$$\frac{[\epsilon(w_1), M] \rightarrow^* [C', M']}{[Q, \epsilon(w_1), M] \xrightarrow{1} [Q, C', M']}$$

3.4 . Type Safety Theorem

In this section, we prove the following type safety theorem.

Theorem 3.4.1. *Both reductions feature subject-reduction and progress.*

3.4.1 . Type safety for circuit-buffering

We extend the type relation to the configuration of the circuit-buffering operational semantics.

Definition 3.4.1 (Extended typing relation for circuit-buffering operational semantics).

$$\frac{(Q, m) \text{ valid} \quad \forall i \text{ leaf} \cdot !\Delta, \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash m_i : A}{!\Delta \vdash (Q, m) : A}$$

Lemma 3.4.2. *Now, we show the following two propositions which say the type judgement preserves:*

- *if $\emptyset \vdash (Q, u) : A$ and $\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \emptyset$, then $\emptyset \vdash (\sigma(Q), \sigma(u)) : A$ and Define the renaming of quantum channel and term, and just check quickly if it holds*

Proof. First, $\vdash (Q, u) : A$ implies that (Q, u) is valid and that $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash u_i : A$. We show that $(\sigma(Q), \sigma(u))$ is valid and that $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(\sigma(Q))_i) \vdash \sigma(u)_i : A$, and hence $\vdash (\sigma(Q), \sigma(u)) : A$. First, notice that $\vdash (Q, u) : A$ implies that $\mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash u_i : A$ at each leaf i , which implies that $\mathbf{FV}(u_i) \subseteq \mathbf{out}(Q)_i$ by lemma 3.2.3. Now, we prove the two premises of the goal as follows:

- If (Q, u) is valid and $\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \emptyset$ then $(\sigma(Q), \sigma(u))$ is valid.

First of all, renaming does not change the shape of the quantum channel Q and the term u . Therefore, it suffices to show that $\sigma(Q)$ is valid and for all leaf i , $\mathbf{out}(\sigma(Q))_i \subseteq \mathbf{FV}(\sigma(u)_i)$. More precisely,

- $\mathbf{st}(Q, V, c)$ then $\mathbf{st}(\sigma(Q), \sigma(V), \sigma(c))$, and
- for each leaf i , if $\mathbf{out}(Q)_i \subseteq \mathbf{FV}(u_i)$, then $\mathbf{out}(\sigma(Q))_i \subseteq \mathbf{FV}(\sigma(u_i))$.

The second follows from that $\mathbf{out}(\sigma(Q))_i = \sigma(\mathbf{out}(Q)_i) \subseteq \sigma(\mathbf{FV}(u_i)) = \mathbf{FV}(\sigma(u_i))$. We prove the first by induction on Q .

- $Q = \epsilon(W)$: We can derive that $\sigma(\epsilon(W)) = \epsilon(\sigma(W))$ and $\mathbf{st}(\epsilon(\sigma(W)), \sigma(W), \sigma(W))$ where $\mathbf{st}(\epsilon(W), W, W)$.
- $Q = U(W) Q'$: First, note that $\sigma(U(W) Q') = U(\sigma(W)) \sigma(Q')$. Then, from the hypothesis, $\mathbf{st}(U(W) Q', V, c)$ we can derive that $W \subseteq V$ and $\mathbf{st}(Q', V, c)$. By applying the induction hypothesis, we obtain that $\mathbf{st}(\sigma(Q'), \sigma(V), \sigma(c))$. Since $\sigma(W) \subseteq \sigma(V)$ follows from $W \subseteq V$, we can derive that $\mathbf{st}(U(\sigma(W)) \sigma(Q'), \sigma(V), \sigma(c))$.
- $Q = \text{init } b \ w \ Q'$: First, note that $\sigma(\text{init } b \ w \ Q') = \text{init } b \ \sigma(w) \ \sigma(Q')$. Then, from the hypothesis, $\mathbf{st}(\text{init } b \ w \ Q', V, c)$ we can derive that $w \notin V$ and $\mathbf{st}(Q', V \cup \{w\}, c)$. By applying the induction hypothesis, we obtain that $\mathbf{st}(\sigma(Q'), \sigma(V) \cup \{\sigma(w)\}, \sigma(c))$. Since $\sigma(w) \notin \sigma(V)$ follows from $w \notin V$ and that σ is a bijective map with $\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \emptyset$, we can derive that $\mathbf{st}(\text{init } b \ \sigma(w) \ \sigma(Q'), \sigma(V), \sigma(c))$.
- $Q = \text{free } w \ Q'$: First, note that $\sigma(\text{free } w \ Q') = \text{free } \sigma(w) \ \sigma(Q')$. Then, from the hypothesis, $\mathbf{st}(\text{free } w \ Q', V, c)$ we can derive that $w \in V$ and $\mathbf{st}(Q', V \setminus \{w\}, c)$. By applying the induction hypothesis, we obtain that $\mathbf{st}(\sigma(Q'), \sigma(V) \setminus \{\sigma(w)\}, \sigma(c))$ (note that $\sigma(V \setminus \{w\}) = \sigma(V) \setminus \{\sigma(w)\}$ since σ is a bijective map with $\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \emptyset$). Since $\sigma(w) \in \sigma(V)$ follows from $w \in V$, we can derive that $\mathbf{st}(\text{free } \sigma(w) \ \sigma(Q'), \sigma(V), \sigma(c))$.
- $Q = \text{meas } w \ Q_1 \ Q_2$: First, note that $\sigma(\text{meas } w \ Q_1 \ Q_2) = \text{meas } \sigma(w) \ \sigma(Q_1) \ \sigma(Q_2)$. Then, from the hypothesis, $\mathbf{st}(\text{meas } w \ Q_1 \ Q_2, V, [c_a, c_b])$ we can derive that $w \in V$, $\mathbf{st}(Q_1, V, c_a)$ and $\mathbf{st}(Q_2, V, c_b)$. By applying the induction hypothesis, we obtain that $\mathbf{st}(\sigma(Q_1), \sigma(V), \sigma(c_a))$ and $\mathbf{st}(\sigma(Q_2), \sigma(V), \sigma(c_b))$. Since $\sigma(w) \in \sigma(V)$ follows from $w \in V$, we can derive that

$$\mathbf{st}(\text{meas } \sigma(w) \ \sigma(Q_1) \ \sigma(Q_2), \sigma(V), [\sigma(c_a), \sigma(c_b)]).$$

- If $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash u_i : A$ and $\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \emptyset$ then $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(\sigma(Q))_i) \vdash \sigma(u_i) : A$.

First, since $\sigma(u)_i = \sigma(u_i)$, $\mathbf{out}(\sigma(Q))_i = \sigma(\mathbf{out}(Q)_i)$, and $\mathbf{FV}(\sigma(u_i)) = \sigma(\mathbf{FV}(u_i))$, it suffices to show that $\mathbf{TC}_Q(\sigma(\mathbf{out}(Q)_i)) \vdash \sigma(u_i) : A$ for each leaf i .

Moreover, from $\mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash u_i : A$, it follows that $\mathbf{FV}(u_i) = \mathbf{out}(Q)_i$ and, hence, $\sigma(\mathbf{out}(Q)_i) = \sigma(\mathbf{FV}(u_i))$.

Therefore, it reduces to show that $\mathbf{TC}_Q(\sigma(\mathbf{FV}(u_i))) \vdash \sigma(u_i) : A$ for each leaf i . Given that $\mathbf{TC}_Q(\mathbf{FV}(u_i)) \vdash u_i : A$, it is natural to accept that $\mathbf{TC}_Q(\sigma(\mathbf{FV}(u_i))) \vdash \sigma(u_i) : A$ is derived from the same type derivation from the one for $\mathbf{TC}_Q(\mathbf{FV}(u_i)) \vdash u_i : A$.

□

In fact, the type judgment of the abstract state is equivalent to the \mathbf{vBind} which was already introduced.

Lemma 3.4.3. *If $\mathbf{shape}(Q) = \mathbf{shape}(u)$, then $\mathbf{vBind}(\epsilon, \mathbf{out}(Q), u, A) \iff \forall i \text{ leaf. } \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash u_i : A$.*

Proof. by induction on u as follows:

- $u = M$: (\Rightarrow) We know that $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), u, A)$ is derived by using the rule \mathbf{vBind}_{nb} , which implies that $\emptyset, \mathbf{TC}_Q(\mathbf{out}(Q)) \vdash u : A$. (\Leftarrow) Since $\emptyset \cap \mathbf{out}(Q) = \emptyset$ and $\mathbf{TC}_Q(\mathbf{out}(Q)) \vdash M : A$, it follows by \mathbf{vBind}_{nb} that $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), M, A)$.
- $u = [u_a, u_b]$: (\Rightarrow) In this case, we know that $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), u, A)$ is derived by \mathbf{vBind}_b rules and $\mathbf{out}(Q) = [c_1, c_2]$. From the induction hypothesis for u_a and u_b , since the rule \mathbf{vBind}_b implies that $\mathbf{vBind}(\emptyset, c_1, u_a, A)$ and $\mathbf{vBind}(\emptyset, c_2, u_b, A)$, we obtain the goal. (\Leftarrow) From the hypothesis, we can derive that $\forall i \text{ leaf. } \mathbf{TC}_Q(c_1) \vdash u_a : A$ and $\forall i \text{ leaf. } \mathbf{TC}_Q(c_2) \vdash u_b : A$. Then by induction hypothesis, we can derive that $\mathbf{vBind}(\emptyset, c_1, u_a, A)$ and $\mathbf{vBind}(\emptyset, c_2, u_b, A)$, and hence $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), u, A)$.

□

Note that the circuit-buffering reduction is terminating.

Lemma 3.4.4 (Termination). *Given a well-typed configuration $\vdash (Q, m) : A$, any reduction sequence starting with (Q, m) is terminating.* □

Subject reduction lemma

We prove the subject reduction lemma for circuit-buffering operational semantics as follows.

Lemma 3.4.5. *If $(Q_1, m_1) \rightarrow (Q_2, m_2)$ and $\vdash (Q_1, m_1) : A$, then $\vdash (Q_2, m_2) : A$.*

Proof. Induction on the reduction relation.

Reduction rules for classical computation

- $(\epsilon(W), (\lambda x.M)V) \rightarrow (\epsilon(W), M[V/x])$

From the hypothesis $\vdash (\epsilon(W), (\lambda x.M)V) : A$, we derive that

$$\mathbf{TC}_Q(W) \vdash (\lambda x.M)V : A.$$

Note that

$$\mathbf{TC}_Q(W) = Q_a \cup Q_b,$$

disjoint union, with

$$Q_a = \mathbf{TC}_Q(\mathbf{FV}((\lambda x.M)) \cap W) \quad \text{and} \quad Q_b = \mathbf{TC}_Q(\mathbf{FV}(V) \cap W).$$

Inverting the typing rule (\rightarrow_E) and (\rightarrow_I) , it can be deduced that $(Q_a, (x : A_t) \vdash M : A)$ and $(Q_b \vdash V : A_t)$ as follows.

$$\frac{\frac{Q_a, (x : A_t) \vdash M : A}{Q_a \vdash \lambda x.M : A_t \multimap A} \quad Q_b \vdash V : A_t}{Q_a, Q_b \vdash (\lambda x.M)V : A}$$

Hence, with Lemma 3.2.5 we infer that

$$Q_a, Q_b \vdash M[V/x] : A.$$

To conclude, we only to remark that $(\epsilon(W), M[V/x])$ is valid.

- $(\epsilon(W), \mathbf{let} \langle x, y \rangle = \langle V, U \rangle \mathbf{in} M) \rightarrow (\epsilon(W), M[V/x, U/y])$

From the hypothesis $\vdash (\epsilon(W), \mathbf{let} \langle x, y \rangle = \langle V, U \rangle \mathbf{in} M)$, we derive that

$$\mathbf{TC}_Q(W) \vdash \mathbf{let} \langle x, y \rangle = \langle V, U \rangle \mathbf{in} M : A$$

Note that

$$\mathbf{TC}_Q(W) = Q_a \cup Q_b,$$

where Q_a and Q_b are disjoint with

$$Q_a = \mathbf{TC}_Q(\mathbf{FV}(\langle V, U \rangle) \cap W) = \mathbf{TC}_Q(\mathbf{FV}(\langle V, U \rangle) \cap W) \quad \text{and} \quad Q_b = \mathbf{TC}_Q((\mathbf{FV}(M) \setminus \{x, y\}) \cap W).$$

Inverting the typing rule (\otimes_E) and (\otimes_I) , it can be deduced that $Q_{a_1} \vdash V : A_a$, $Q_{a_2} \vdash U : A_b$, and $Q_b, (x : A_a), (y : A_b) \vdash M : A$ as follows.

$$\frac{\frac{Q_{a_1} \vdash V : A_a \quad Q_{a_2} \vdash U : A_b}{Q_a \vdash \langle V, U \rangle : A_a \otimes A_b} \quad Q_b, (x : A_a), (y : A_b) \vdash M : A}{Q_a, Q_b \vdash \mathbf{let} \langle x, y \rangle = \langle V, U \rangle \mathbf{in} M : A}$$

Hence, with Lemma 3.2.5 we infer that

$$Q_{a_1}, Q_b \vdash M[V/x] : A, \quad Q_{a_2}, Q_b \vdash M[U/y] : A, \quad \text{and} \quad Q_a, Q_b \vdash M[V/x, U/y].$$

To conclude, we remark that $(\epsilon(W), M[V/x, U/y])$ is valid.

- $(\epsilon(W), \mathbf{if\ tt\ then\ } M_a \mathbf{\ else\ } M_b) \rightarrow (\epsilon(W), M_a)$

From the hypothesis $\vdash (\epsilon(W), \mathbf{if\ tt\ then\ } M_a \mathbf{\ else\ } M_b) : A$, we derive that

$$\mathbf{TC}_Q(W) \vdash \mathbf{if\ tt\ then\ } M_a \mathbf{\ else\ } M_b : A$$

Note that

$$\mathbf{TC}_Q(W) = Q_a \cup Q_b$$

with disjoint union of

$$Q_a = \mathbf{TC}_Q(\mathbf{FV}(M_a) \cap W) \text{ and } Q_b = \mathbf{TC}_Q(\mathbf{FV}(M_b) \cap W).$$

Inverting the typing rule (if), it can be deduced that $Q_a \vdash M_a : A$ and $Q_b \vdash M_b : A$ as follows.

$$\frac{\vdash \mathbf{tt} : \mathbf{bool} \quad Q_a \vdash M_a : A \quad Q_b \vdash M_b : A}{Q_a, Q_b \vdash \mathbf{if\ tt\ then\ } M_a \mathbf{\ else\ } M_b : A}$$

Hence, we can conclude that

$$\vdash (\epsilon(W), M_a) : A$$

since $(\epsilon(W), M_a)$ is valid.

- $(\epsilon(W), \mathbf{if\ ff\ then\ } M_a \mathbf{\ else\ } M_b) \rightarrow (\epsilon(W), M_b)$

Similar to the previous case.

Reduction rules for circuit operation

- $(\epsilon(\emptyset), \mathbf{box\ } V) \rightarrow (\epsilon(\emptyset), (p, \epsilon(W_p), Vp))$
where $p = \mathbf{new}(P)$ and $W_p = \mathbf{supp}(p)$.

From the hypothesis $(\vdash (\epsilon(W), \mathbf{box\ } V) : A_t)$, it can be deduced that $A_t = !\mathbf{QChan}(P, A)$ and that

$$\mathbf{TC}_Q(\emptyset) \vdash \mathbf{box\ } V : !\mathbf{QChan}(P, A). \quad (3.8)$$

By unfolding the available typing rules and simplifying, we can derive that

$$\vdash V : !(P \multimap A). \quad (3.9)$$

From Lemma 3.3.2, since $p = \mathbf{new}(P)$, it follows that $p \vDash P$ and that $(\mathbf{TC}_Q(W_p) \vdash p : P)$.

$$\mathbf{TC}_Q(W_p) = \mathbf{TC}_Q(\mathbf{FV}(Vp) \cap W_p).$$

Consequently, we can conclude that $(\vdash (\epsilon(\emptyset), (p, \epsilon(W_p), Vp)) : !\text{QChan}(P, A))$ since

$$\frac{p \vDash P \quad \frac{W_p \cap \emptyset = \emptyset \quad \frac{\vdash V : !(P \multimap A) \quad \vdash V : P \multimap A}{\text{TC}_Q(W_p) \vdash p : P}}{\text{TC}_Q(W_p) \vdash Vp : A}}{\mathbf{vBind}(\emptyset, W_p, Vp, A)} \quad \vdash (p, \epsilon(W_p), Vp) : !\text{QChan}(P, A)$$

and since $(\epsilon(\emptyset), (p, \epsilon(W_p), Vp))$ is trivially a valid configuration.

- $(\epsilon(\mathbf{FV}(V)), (\text{unbox}(p, Q, u))V) \rightarrow (\sigma(Q), \sigma(u))$ where $\mathbf{shape}(p) = \mathbf{shape}(V)$ and $\sigma = \mathbf{bind}(p, V)$.

From the hypothesis

$$\vdash (\epsilon(\mathbf{FV}(V)), (\text{unbox}(p, Q, u))V) : A$$

it can be deduced that $(\vdash (p, Q, u) : \text{QChan}(P, A))$, and $(\vdash V : P)$ as follows.

$$\frac{\frac{\frac{\vdash \text{unbox} : \mathbf{QChan}(P, A) \multimap (P \multimap A) \quad p \vDash P \quad \mathbf{vBind}(\emptyset, \mathbf{out}(Q), u, A)}{\vdash (p, Q, u) : !\mathbf{QChan}(P, A)}}{\vdash (p, Q, u) : \mathbf{QChan}(P, A)}}{\vdash \text{unbox}(p, Q, u) : P \multimap A} \quad \text{TC}_Q(\mathbf{FV}(V)) \vdash V : P}{\text{TC}_Q(\mathbf{FV}(V)) \vdash (\text{unbox}(p, Q, u))V : A}$$

From $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), u, A)$ and Lemma 3.4.3, we can derive that $\forall i$ leaf $\text{TC}_Q(\mathbf{out}(Q)_i) \vdash u_i : A$. And, actually, $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), u, A)$ implies that $\mathbf{shape}(Q) = \mathbf{shape}(u)$.

Therefore, we obtain $\emptyset \vdash (Q, u) : A$. (Note that the validity of quantum channel constant (p, Q, u) implies that Q is valid.)

Now, from Lemma 3.4.2, we have the following proposition:

- if $\emptyset \vdash (Q, u) : A$ and $\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \emptyset$, then $\emptyset \vdash (\sigma(Q), \sigma(u)) : A$

Then, we conclude that

$$\emptyset \vdash (\sigma(Q), \sigma(u)) : A$$

since

$$\mathbf{ran}(\sigma) \cap \mathbf{all}(Q) = \mathbf{FV}(V) \cap \mathbf{all}(Q) = \emptyset \cap \mathbf{all}(Q) = \emptyset$$

Structural reduction rule for quantum channel constant

- $(\epsilon(\emptyset), (p, Q, m)) \rightarrow (\epsilon(\emptyset), (p, Q', m'))$

where $(Q, m) \rightarrow (Q', m')$.

By the induction hypothesis, we have, for any type A_a that $(\vdash (Q', m') : A_a)$ if $(\vdash (Q, m) : A_a)$.

From the hypothesis $(\vdash (\epsilon(\emptyset), (p, Q, m)) : A_t)$, it follows that $A_t = !\text{QChan}(P, A)$ and $\vdash (p, Q, m) : A_t$. Moreover, from the typing rule QChan_I , we derive that $p \vDash P$ and $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), m, A)$. Thus, according to Lemma 3.4.3, we can derive that $\mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash m_i : A$ for each leaf i .

Then, since (Q, m) and (Q', m') are valid (from (p, Q, m) and (p, Q', m') and the lemma 3.3.1), it follows that $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash m_i : A$ implies that $\vdash (Q, m) : A$.

Then, from the induction hypothesis, we get that $\vdash (Q', m') : A$, which implies that (Q', m') is valid and that for all leaf i , $\mathbf{TC}_Q(\mathbf{out}(Q')_i) \vdash m'_i : A$. Again by the same lemma (Lemma 3.4.3), we obtain that $\mathbf{vBind}(\emptyset, \mathbf{out}(Q'), m', A)$. Finally, applying the QChan_I rule, we conclude that $\vdash (p, Q', m') : !\text{QChan}(P, A)$.

Therefore, since (Q', m') is valid, we derive $\emptyset \vdash (\epsilon(\emptyset), (p, Q', m')) : !\text{QChan}(P, A)$ from $\mathbf{TC}_Q(\emptyset) \vdash (p, Q', m') : !\text{QChan}(P, A)$.

Structural reduction rules for empty quantum channel

- $(\epsilon(W_M \cup W_N), NM) \rightarrow (\mathbf{extend}(Q, W_N), Nm)$

where $(\epsilon(W_M), M) \rightarrow (Q, m)$ and $\mathbf{all}(Q) \cap W_N = \emptyset$.

From the hypothesis, $\vdash (\epsilon(W_M \cup W_N), NM) : A$, we derive that

$$\mathbf{TC}_Q(W_M \cup W_N) \vdash NM : A.$$

From Lemma 3.2.3, we get that $W_M \cup W_N = \mathbf{FV}(NM)$. Then, we let that $W_M = \mathbf{FV}(M) \setminus \emptyset$ and $W_N = \mathbf{FV}(N) \setminus \emptyset$.

Inverting the typing rule (\multimap_E) , it can be deduced that $(Q_a \vdash N : A_t \multimap A)$ and $(Q_b \vdash M : A_t)$ as follows

$$\frac{Q_a \vdash N : A_t \multimap A \quad Q_b \vdash M : A_t}{Q_a, Q_b \vdash NM : A,}$$

Note that $\mathbf{TC}_Q(W_M \cup W_N)$ is a disjoint union of $Q_a = \mathbf{TC}_Q(W_N)$ and $Q_b = \mathbf{TC}_Q(W_M)$.

We show the following two proposition:

- $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash m_i : A_t$
 Note that $(Q_b \vdash M : A_t)$ implies that $(\vdash (\epsilon(W_M), M) : A_t)$. Then, by the induction hypothesis, we obtain that $(\vdash (Q, m) : A_t)$. It follows that $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash m_i : A_t$.
- $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)_i \cup W_N) \vdash Nm_i : A_t \multimap A$ where $\mathbf{out}(Q)_i \cup W_N = \mathbf{out}(\mathbf{extend}(Q, W_N))_i$ by Lemma 3.1.4
 From $Q_a \vdash N : A_t \multimap A$ and $\mathbf{TC}_Q(\mathbf{out}(Q)_i) \vdash m_i : A_t$, we can derive that $\mathbf{TC}_Q(\mathbf{out}(Q)_i), Q_a \vdash Nm_i : A$ by applying the typing rule (\multimap_E) since $\mathbf{out}(Q)_i \cap W_N \subseteq \mathbf{all}(Q) \cap W_N = \emptyset$.

Finally, since $\mathbf{extend}(Q, W_N)$ is valid (by Lemma 3.1.4) and $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(\mathbf{extend}(Q, W_N))_i) \vdash Nm_i : A_t \multimap A$, we can conclude that $\vdash (\mathbf{extend}(Q, W_N), Nm)$.

- Other structural rules for empty quantum channel
 - $(\epsilon(W_M \cup W_N), MN) \rightarrow (\mathbf{extend}(Q, W_N), mN)$,
 - $(\epsilon(W_M \cup W_N), \langle M, N \rangle) \rightarrow (\mathbf{extend}(Q, W_N), \langle m, N \rangle)$,
 - $(\epsilon(W_M \cup W_N), \langle N, M \rangle) \rightarrow (\mathbf{extend}(Q, W_N), \langle N, m \rangle)$,
 - $(\epsilon(W_M \cup W_N), \mathbf{if } M \mathbf{ then } M_a \mathbf{ else } M_b) \rightarrow$
 $(\mathbf{extend}(Q, W_N), \mathbf{if } m \mathbf{ then } M_a \mathbf{ else } M_b)$,
 - $(\epsilon(W_M \cup W_N), \mathbf{let } \langle x, y \rangle = M \mathbf{ in } N) \rightarrow$
 $(\mathbf{extend}(Q, W_N), \mathbf{let } \langle x, y \rangle = m \mathbf{ in } N)$,

where $(\epsilon(W_M), M) \rightarrow (Q, m)$ and $\mathbf{all}(Q) \cap W_N = \emptyset$, can be shown similarly.

Structural reduction rules for non-empty quantum channel

- $((\text{meas } w \ Q_1 \ Q_2), [m_a, m_b]) \rightarrow ((\text{meas } w \ Q_3 \ Q_4), [m_c, m_d])$
 where $(Q_1, m_a) \rightarrow (Q_3, m_c)$ and $(Q_2, m_b) \rightarrow (Q_4, m_d)$.
 From the hypothesis $(\vdash ((\text{meas } w \ Q_1 \ Q_2), [m_a, m_b]) : A)$, it follows that $(\vdash (Q_1, m_a) : A)$ and $(\vdash (Q_2, m_b) : A)$.
 Then, by the induction hypothesis, we obtain $(\vdash (Q_3, m_c) : A)$ and $(\vdash (Q_4, m_d) : A)$.
 Therefore, we can conclude that $(\vdash ((\text{meas } w \ Q_3 \ Q_4), [m_c, m_d]) : A)$.
- $((\text{meas } w \ Q_1 \ Q_2), [m_a, v]) \rightarrow ((\text{meas } w \ Q_3 \ Q_2), [m_c, v])$
 where $(Q_1, m_a) \rightarrow (Q_3, m_c)$.
 Similar to the previous case.

- $((\text{meas } w \ Q_1 \ Q_2), [v, m_b]) \rightarrow ((\text{meas } w \ Q_1 \ Q_4, [v, m_d])$

where $(Q_2, m_b) \rightarrow (Q_4, m_d)$.

Similar to the previous case.

- $(Q'_1, m_a) \rightarrow (Q'_2, m_b)$

where $(Q_1, m_a) \rightarrow (Q_2, m_b)$.

Note that Q'_1 can be either $U(W) \ Q_1$, $\text{init } b_0 \ w \ Q_1$, and $\text{free } w \ Q_1$. Each case can be proved similarly to the previous case using induction hypothesis.

□

Progress lemma

The progress lemma states that all well-typed terms in a state either reduce to another term or is a value. It consists of, first, showing it for the non-branching term whose type does not contain QChan , and second,

Lemma 3.4.6. *Progress lemma consists of the following three propositions:*

- *if $\vdash (\epsilon(W), M) : A$, then either there exists (Q, m) such that $(\epsilon(W), M) \rightarrow (Q, m)$ or M is a value, and*
- *if $\vdash (Q_1, m_1) : A$, then either there exists (Q_2, m_2) such that $(Q_1, m_1) \rightarrow (Q_2, m_2)$ or m_1 is a value*
- *if **valid** (Q, m) and **vBind** $(\emptyset, \text{out}(Q), m, A)$ then either m is a value or (Q, m) reduces,*

Proof. We prove the two propositions by induction on the level of QChan of m, M and m_1 .

For the basis case: the QChan level of m_1 is -1 . The propositions are true since there is no term with the QChan level -1 .

For the induction step: the proof follows.

1. if $\vdash (\epsilon(W), M) : A$, then either there exists (Q, m) such that $(\epsilon(W), M) \rightarrow (Q, m)$ or M is a value

Note that $\vdash (\epsilon(W), M) : A$ is equivalent to that $\mathbf{TC}_Q(W) \vdash M : A$. Therefore, it suffices to show that if $\mathbf{TC}_Q(W) \vdash M : A$, then either there exists (Q, m) such that $(\epsilon(W), M) \rightarrow (Q, m)$ or M is a value. Note that $W = \mathbf{FV}(M) \setminus \emptyset$. We prove it by induction on the type derivation $\mathbf{TC}_Q(W) \vdash M : A$.

- $(x : A \vdash x : A), (\vdash * : I), (\vdash \text{tt} : \text{bool}), (\vdash \text{ff} : \text{bool}), (\vdash \text{box} : !(P \multimap A) \multimap !\text{QChan}(P, A)),$ and $\vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A)$

In these cases, it can be checked that M is value and the configuration $(\epsilon(W), M)$ does not reduce.

- $(\mathbf{TC}_Q(W) \vdash M : A)$ where $(\vdash M : !A)$
Directly follows from the induction hypothesis.
- $(\vdash V : !A)$ where $(\vdash V : A)$ and V is a value
Directly follows from the induction hypothesis.
- $(Q \vdash \lambda x.M : A_a \multimap A_b)$ where $(Q, (x : A_a) \vdash M : A_b)$.
 $\lambda x.M$ is a value and there is no reduction rule applicable for the term.
- $(Q_a, Q_b \vdash M_a M_b : A_b)$ where $(Q_a \vdash M_a : A_a \multimap A_b)$ and $(Q_b \vdash M_b : A_a)$.

We show the goal by case analysis on M_a and M_b .

M_a **is a value**: Again we do case analysis on M_b .

- M_b **is a value**: It follows from $\Gamma_a \vdash M_a : A_a \multimap A_b$ that M_a is either $\lambda x.M$, **box**, or **unbox**.

$M_a = \lambda x.M$: First, $\lambda x.M M_b$ is not a value. Moreover, $(\epsilon(W), (\lambda x.M)M_b)$ reduces to $(\epsilon(W), M[M_b/x])$ as follows.

$$\overline{(\epsilon(W), (\lambda x.M)M_b) \rightarrow (\epsilon(W), M[M_b/x])}$$

$M_a = \mathbf{box}$: First, $\mathbf{box}M_b$ is not a value. Reduction rules for **box** reduces the configuration since M_b is a value.

$M_a = \mathbf{unbox}$: In this case, $\mathbf{unbox}M_b$ is a value since M_b is a value. Moreover, it can be checked that there is not rule to reduce $\mathbf{unbox}M_b$.

- M_b **is not a value**: First of all, $M_a M_b$ is not a value. Next, by induction hypothesis, $(\epsilon(W_M), M_b) \rightarrow (Q, m_b)$ for some Q and m_b . (Note that $Q_b = \mathbf{TC}_Q(W_M)$.) Based on the equality of the configuration, we can assume that $\mathbf{all}(Q) \cap W_N = \emptyset$, i.e. there exists some σ such that $(Q, m_b) \sim (\sigma(Q), \sigma(m_b))$. Since the reduction is defined over the equivalence class of the configuration, we can say that $(\epsilon(W_M), M_b) \rightarrow (\sigma(Q), \sigma(m_b))$. Then, we can obtain $(\epsilon(W), M_a M_b) \rightarrow (\mathbf{extend}(\sigma(Q), W_N), M_a m_b)$.

M_a **is not a value**: Similarly, $M_a M_b$ is not a value and by induction hypothesis, $(\epsilon(W), M_a) \rightarrow (Q, m_a) \sim (\sigma(Q), \sigma(m_a))$ where $\mathbf{all}(\sigma(Q)) \cap W_N = \emptyset$. Therefore, we can obtain $(\epsilon(W), M_a M_b) \rightarrow (\mathbf{extend}(\sigma(Q), W_N), \sigma(m_a)M_b)$.

- $(Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b)$ where $(Q_a \vdash M_a : A_a)$ and $(Q_b \vdash M_b : A_b)$.

We do the case analysis on M_a and M_b .

Both M_a and M_b are values: It follows that $\langle M_a, M_b \rangle$ is a value. Moreover, a proof by contradiction shows that there is no applicable reduction rule for the configuration, $(\epsilon(W), \langle M_a, M_b \rangle)$.

M_a or M_b is not a value: In this case, $\langle M_a, M_b \rangle$ is not a value. We can also find a reduction for $(\epsilon(W), \langle M_a, M_b \rangle)$ by applying the structural rules for pair.

- $(Q_a, Q_b \vdash \mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b : A)$ where $(Q_a \vdash M_a : A_a \otimes A_b)$ and $(Q_b, (x : A_a), (y : A_b) \vdash M_b : A)$.

Since $(\mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b)$ is not a value, we show that we can find a reduction for the configuration $(\epsilon(W), \mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b)$. We show it by case analysis on M_a .

M_a is a value: It follows that $M_a = \langle V, U \rangle$ for some values V and U . (Note that M_a cannot be a variable since Q_a and Q_b is list of qubits.) Then $(\epsilon(W), \mathbf{let} \langle x, y \rangle = \langle V, U \rangle \mathbf{in} M) \rightarrow (\epsilon(W), M[V/x, U/y])$.

M_a is not a value:

By induction hypothesis, the configuration $(\epsilon(W_M), M_a)$ reduces to some configuration $(Q, m_a) \sim (\sigma(Q), \sigma(m_a))$ where $W_M = \mathbf{FV}(M_a) \setminus \emptyset$ and $\mathbf{all}(\sigma(Q)) \cap (W \setminus W_M) = \emptyset$. Then by applying the structural reduction rule for let, we obtain that $(\epsilon(W), \mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b) \rightarrow (\mathbf{extend}(\sigma(Q), W_N), \mathbf{let} \langle x, y \rangle = \sigma(m_a) \mathbf{in} M_b)$.

- $(Q_a, Q_b \vdash \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b : A)$ where $(Q_a \vdash M : \mathbf{bool})$, $(Q_b \vdash M_a : A)$, and $(Q_b \vdash M_b : A)$.

It can be shown similarly to the previous case that $(\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b)$ is not a value and the configuration $(\epsilon(W), \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b : A)$ reduces to another configuration by case analysis on M .

M is a value: It follows that M is tt or ff. (Note that M cannot be a variable since Q_a and Q_b is list of qubits which does not include boolean.) Then $(\epsilon(W), \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b)$ reduces to either $(\epsilon(W), M_a)$ or $(\epsilon(W), M_b)$ depending on M .

M is not a value:

By induction hypothesis, the configuration $(\epsilon(W_M), M)$ reduces to some configuration $(Q, m) \sim (\sigma(Q), \sigma(M))$ where $W_M = \mathbf{FV}(M) \setminus \emptyset$ and $\mathbf{all}(\sigma(Q)) \cap (W \setminus W_M) = \emptyset$. Then by applying the structural rule for if, we obtain that $(\epsilon(W), \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b) \rightarrow (\mathbf{extend}(\sigma(Q), (W \setminus W_N)), \mathbf{if} \sigma(m) \mathbf{then} M_a \mathbf{else} M_b)$.

- $(\vdash (p, Q, m) : !\mathbf{QChan}(P, A))$ where $(p \vDash P)$ and $(\mathbf{vBind}(\emptyset, \mathbf{out}(Q), m, A))$.

m is a value: It follows that the quantum channel constant (p, Q, m) is a value. Hence, it suffices to show that there is no reduction rule to applicable to the configuration $(\epsilon(\emptyset), (p, Q, m))$. In fact, the only reduction rule regarding quantum channel constants requires that the configuration (Q, m) reduces. However, it contradicts the induction hypothesis.

m is not a value: Notice that quantum channel level of $\text{QChan}(P, A)$ is strictly greater than A . Therefore, we use the induction hypothesis on the third proposition: if $\mathbf{valid}(Q, m)$ and $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), m, A)$ then either m is a value or (Q, m) reduces (note that $\mathbf{valid}(Q, m)$ follows from Lemma 3.3.1 for (p, Q, m)). Since m is not a value, it follows that (Q, m) reduces to, say, (Q', m') . Therefore, by applying the reduction rule on quantum channel, we obtain that $(\epsilon(\emptyset), (p, Q, m)) \rightarrow (\epsilon(\emptyset), (p, Q', m'))$.

- $(\gamma_a \times \gamma_b \vdash [m_a, m_b] : A)$ where $(\gamma_a \vdash m_a : A)$ and $(\gamma_b \vdash m_b : A)$. Since $(\gamma_a \times \gamma_b)$ is a branching typing context and $[m_a, m_b]$ is a branching term, it contradicts the hypothesis.

2. if $(\vdash (Q_1, m_1) : A)$, then either there exists (Q_2, m_2) such that $(Q_1, m_1) \rightarrow (Q_2, m_2)$ or m_1 is a value

Note that $\vdash (Q_1, m_1) : A$ implies that (Q_1, m_1) is valid. So let us prove it by induction on the validity of the circuit-buffering configuration (Q_1, m_1) .

- $(\epsilon(W), M)$
This case is shown by the previous proposition.
- $(U(W) Q, m)$ where (Q, m)
By induction hypothesis, we know that either there exists (Q', m') such that $(Q, m) \rightarrow (Q', m')$ or m is a value.
Suppose there exists some (Q', m') such that $(Q, m) \rightarrow (Q', m')$. Then by the reduction, we obtain that $(U(W) Q, m) \rightarrow (U(W) Q', m')$. Also, we know that m is not a value in this case.
Otherwise, we cannot apply the reduction rule for the circuit, and it follows that m is value.
- $(\text{init } b \ w \ Q, m)$ and $(\text{free } w \ Q, m)$, where (Q, m)
Shown similarly to the previous case.
- $(\text{meas } w \ Q_a \ Q_b, [m_a, m_b])$ where (Q_a, m_a) and (Q_b, m_b)
Both m_a and m_b are values: It follows that $[m_a, m_b]$ is a value. Suppose that $(\text{meas } w \ Q_a \ Q_b, [m_a, m_b])$ reduces to another configuration. Then by the operational semantics, we can deduce that either (Q_a, m_a) or (Q_b, m_b) , or both, reduces to another state. However, by the induction hypothesis, none of them is reducible.
Either m_a or m_b is a value: It follows that $[m_a, m_b]$ is not a value. Since either (Q_a, m_a) or (Q_b, m_b) is reducible, the configuration $(\text{meas } w \ Q_a \ Q_b, [m_a, m_b])$ is reducible.
Neither m_a nor m_b is a value: It follows that $[m_a, m_b]$ is not a value and similarly to the previous case, we can find a reduction for the configuration $(\text{meas } w \ Q_a \ Q_b, [m_a, m_b])$.

3. if $\mathbf{valid}(Q, m)$ and $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), m, A)$ then either m is a value or (Q, m) reduces

First, we prove that $\mathbf{valid}(Q, m)$ and $\mathbf{vBind}(\emptyset, \mathbf{out}(Q), m, A)$ implies that $\vdash (Q, m) : A$. Then the goal follows from the second proposition: if $\vdash (Q, m) : A$ then either there exists (Q_2, m_2) such that $(Q, m) \rightarrow (Q_2, m_2)$ or m is a value.

We prove by induction on Q .

- $Q = \epsilon(W)$
From $\mathbf{valid}(\epsilon(W), m)$, we know that $m = M$ a non-branching term. Then, $\mathbf{vBind}(\emptyset, \mathbf{out}(\epsilon(W)), m, A)$ implies that $\mathbf{TC}_Q(W) \vdash M : A$. Therefore, we can obtain $\vdash (\epsilon(W), M) : A$ since we know that (Q, m) is valid.
- $Q = U(W) Q'$, (init b w Q, m), and (free w Q, m)
From the induction hypothesis, since $\mathbf{valid}(Q', m)$ (which is obtained from $\mathbf{valid}(Q, m)$) and $\mathbf{out}(Q) = \mathbf{out}(Q')$, we can obtain that $\vdash (Q', m) : A$. It implies $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q')_i) \vdash m_i : A$, hence $\vdash (Q, m) : A$.
- $Q = \text{meas } w Q_a Q_b$
First of all, we can obtain that $\mathbf{valid}(Q_a, m_a)$ and $\mathbf{valid}(Q_b, m_b)$ (from $\mathbf{valid}(Q, m)$) where $m = [m_a, m_b]$.
Moreover, $\mathbf{vBind}(\emptyset, [\mathbf{out}(Q_a), \mathbf{out}(Q_b)], [m_a, m_b], A)$ implies $\mathbf{vBind}(\emptyset, \mathbf{out}(Q_a), m_a, A)$ and $\mathbf{vBind}(\emptyset, \mathbf{out}(Q_b), m_b, A)$. Therefore, from the induction hypothesis, we obtain that $\vdash (Q_a, m_a) : A$ and $\vdash (Q_b, m_b) : A$. These again imply that $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q_a)_i) \vdash m_{a_i} : A$ and $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q_b)_i) \vdash m_{b_i} : A$. Hence, we can conclude $\forall i \text{ leaf} \cdot \mathbf{TC}_Q(\mathbf{out}(Q)) \vdash m_i : A$, and $\vdash (Q, m) : A$.

□

3.4.2 . Type safety for QRAM

We extend the type relation to the configuration of the QRAM-based operational semantics.

Definition 3.4.2 (Extended typing relation for QRAM-based operational semantics).

$$\frac{!\Delta \vdash (Q, m) : A}{!\Delta \vdash (\phi, L, Q, m) : A}$$

Subject reduction lemma

Lemma 3.4.7. *If $\vdash (\phi_1, L_1, Q_1, m_a) : A$ and $(\phi_1, L_1, Q_1, m_a) \xrightarrow{p} (\phi_2, L_2, Q_2, m_b)$, then $\vdash (\phi_2, L_2, Q_2, m_b) : A$.*

Proof. From the hypothesis $(\vdash (\phi_1, L_1, Q_1, m_a) : A)$, we can derive that $(\vdash (Q_1, m_a) : A)$ as follows.

$$\frac{\vdash (Q_1, m_a) : A}{\vdash (\phi_1, L_1, Q_1, m_a) : A}$$

We prove by induction on the QRAM-based reduction relation, $(\phi_1, L_1, Q_1, m_a) \xrightarrow{p} (\phi_2, L_2, Q_2, m_b)$.

- $(\phi_1, L_1, Q_1, m) \xrightarrow{1} (\phi_2, L_2, Q_2, m)$, where $(Q_1 = U(W) Q_2)$ and $(\phi_2, L_2) = \text{Unitary}(U, W, \phi_1, L_1)$.

It suffices to show that $(\vdash (Q_2, m) : A)$ since

$$\frac{\vdash (Q_2, m) : A}{\vdash (\phi_2, L_2, Q_2, m) : A}$$

However, it follows from the fact that $(\vdash (Q_1, m_a) : A)$ by definition as follows

$$\frac{\vdash (Q_2, m) : A}{\vdash (U(W) Q_2, m_a) : A}$$

- $(\phi_1, L_1, Q_1, m) \xrightarrow{1} (\phi_2, L_2, Q_2, m)$, where $(Q_1 = \text{init } b_0 \ w \ Q_2)$ and $(\phi_2, L_2) = \text{Init}(b_0, w, \phi_1, L_1)$.

Similar to the previous case.

- $(\phi_1, L_1, Q_1, [m_t, m_f]) \xrightarrow{p_x} (\phi_x, L_x, Q_x, m_x)$, where $(Q_1 = \text{meas } w \ Q_t \ Q_f)$ and $(\phi_t, L_t, p_t, \phi_f, L_f, p_f) = \text{Meas}(w, \phi_1, L_1)$.

Similar to the previous case.

- $(\phi_1, L_1, Q_1, m) \xrightarrow{p_x} (\phi_x, L_x, Q, m)$, where $(Q_1 = \text{free } w \ Q)$ and $(\phi_t, L_t, p_t, \phi_f, L_f, p_f) = \text{Free}(w, \phi_1, L_1)$.

Similar to the previous case.

- $(\phi, L, \epsilon(W), M) \xrightarrow{1} (\phi, L, Q, m)$, where $(\epsilon(W), M) \rightarrow^* (Q, m)$.

It suffices to show that $(Q, m) : A$ since

$$\frac{\vdash (Q, m) : A}{\vdash (\phi, L, Q, m) : A}$$

However, by lemma 3.4.5, we can derive that $(\vdash (Q, m) : A)$ as follows.

$$\frac{\vdash (\epsilon(W), M) : A \quad (\epsilon(W), M) \rightarrow^* (Q, m)}{\vdash (Q, m) : A}$$

□

Progress lemma

Lemma 3.4.8. *If $(! \Delta \vdash (\phi_1, L_1, Q_1, m_a) : A)$, then either there exists (ϕ_2, L_2, Q_2, m_b) such that $(\phi_1, L_1, Q_1, m_a) \xrightarrow{p} (\phi_2, L_2, Q_2, m_b)$, or $(Q_1 = \epsilon(W))$ and m_a is a value.*

Proof. We prove by case analysis on Q_1 .

If $Q_1 = \epsilon(W)$:

It follows from $(\vdash (\phi, L, \epsilon(W), M) : A)$ that $(\vdash (\epsilon(W), M) : A)$. Then, by lemma 3.4.6, it follows that either M is a value or there is some (Q_t, m_t) such that $(\epsilon(W), M) \rightarrow (Q_t, m_t)$. If M is not a value, it follows that $(\phi, L, \epsilon(W), M) \xrightarrow{p} (\phi, L, Q_t, m_t)$ as follows.

$$\frac{(\epsilon(W), M) \rightarrow (Q_t, m_t)}{(\phi, L, \epsilon(W), M) \xrightarrow{1} (\phi, L, Q_t, m_t)}$$

If M is a value, then $(\epsilon(W), M)$ is not reducible, hence, we cannot apply the reduction rule to the configuration $(\phi, L, \epsilon(W), M)$.

Otherwise:

We can find for each case (where Q_1 is one of $(U(W) Q)$, $(\text{init } b \ w \ Q)$, $(\text{free } w \ Q)$, and $(\text{meas } w \ Q_1 \ Q_2)$) the configuration (ϕ_1, L_1, Q_1, m_a) reduces to some other configuration.

□

4 - Categorical Model

Categorical semantics of programming languages and type systems is based on the interpretation of each typing rule in typing derivation, which maps type to object and typing judgment to a morphism of a certain category. For example, categorical semantics of simply-typed λ -calculus is based on the cartesian closed category, which is a category with a terminal object, a product, and an exponential. In the semantics, a typing context is interpreted as a product of the type assigned to each variable, and a type judgment is mapped to a morphism from the object of the typing context to the object of the type assigned to the term.

Concerning quantum programming language, one needs to distinguish classical variables from quantum variables since quantum data (qubits) cannot be duplicated or erased while classical data (bits or functions) can be used as many times as wanted. The difference is usually handled by an indicator called bang $!$, which indicates that the type is classical, as in the linear logic. Note that one can safely regard a classical type as a linear type, which corresponds to the dereliction of linear logic, whereas one can promote a type only if the type does not contain quantum data.

On the side of categorical semantics, Benton's linear/non-linear model is used to interpret the typing rules for dereliction and promotion. Briefly, the model consists of a symmetric monoidal adjunction between a cartesian closed category \mathcal{C} and a symmetric monoidal closed category \mathcal{L} , which models the non-linear and linear type systems, respectively. The adjunction gives a comonad $(!, \epsilon, \delta)$ where $!$ is a functor $\mathcal{L} \rightarrow \mathcal{C}$, $\epsilon : ! \rightarrow \mathbf{id}$ is the counit, and $\delta : ! \rightarrow !!$ is the comultiplication. The model allows us to interpret the dereliction rule as applying the counit and the promotion rule by using the comultiplication.

Once a symmetric monoidal closed category from the linear/non-linear model is introduced to the model, one can interpret the circuit type $\text{Circ}(A, B)$ as the internal hom $(-\multimap)$ of the object corresponding to the type A and B . Moreover, the monoidal adjunction enables us to define the box and unbox circuit operators. Therefore, the linear/non-linear model can be used for the categorical semantics of quantum circuit description languages. However, since both A and B are objects in the symmetric monoidal closed category, they need to be a linear type. In particular, it is not clear how to interpret a non-trivial quantum channel with measurement, whose type would be $\text{QChan}(\text{qubit}, \text{bit})$ where bit represents the binary classical control flow which can be distinguished from the Boolean type since bit-type term needs a simulation (which is probabilistic computation) on quantum co-processor to obtain a Boolean value.

To model the classical control flow in the quantum channel, one needs a particular symmetric monoidal closed category with extra properties, like the coproduct completion of a symmetric monoidal closed category in [55] or fibration of a symmetric monoidal closed category over a locally cartesian closed category as the model of the linear dependent type theory in [24]. The coproduct completion construction in the first model is an example of the state-parameter fibration from the latter. In this example, the objects represent functions from the classical types to quantum types, which lets us define the family of quantum circuits or the parameterized quantum circuits. Therefore, more general types like $\text{bit} = I + I$ (defined by using the coproduct) can be defined in the completion of the monoidal category. In addition, since the category is again symmetric monoidal closed, one can define the quantum channel type $\text{QChan}(A, B)$ as the internal hom or a parameter object corresponding to the homset $\text{Hom}(A, B)$. Therefore, in their model, one can interpret higher-order circuit-description languages, and several extensions of the semantics [24, 38] have been discussed. However, none of them were explicitly shown to capture dynamic lifting: the possibility to change behavior depending on the result of a measurement.

Our model for the language and dynamic lifting relies on this coproduct completion model in [55]. As we can consider a classical control flow as a tree, which can be represented as a set of the prefix, the configuration of the quantum channel and the term can be interpreted as a family of parameterized circuits, where each leaf corresponds to a parameterized circuit. Based on Moggi’s interpretation of side effect [44], we define a monad (F, μ, η) to model the non-deterministic computational effect of the creation of branches: the unit μ interprets the program without branching effect and the multiplication η transforms a family of parameterized circuits into a parameterized circuit. Then the type judgment is interpreted as a morphism in the Kleisli category, and the quantum channel type $\text{QChan}(A, B)$ is interpreted as the parameter object from the homset $\text{Hom}(A, B)$ in the Kleisli category.

Moreover, our model is based on a concrete category of diagrams which allows us to interpret quantum programs written in the language into a collection of diagrams. Diagrams are used in many places to represent different logic (proof net for linear logic) and models of computation (like the quantum circuit for quantum computation). A good example is ZX-calculus which has been shown to be able to represent all quantum processes. Moreover, the complete equivalence theory of ZX-calculus allows us to reason the equality between two quantum processes. In line with those diagrams, our diagram represents the quantum process with measurement and comes with an equivalence theory of diagrams, which is necessary to construct the entire categorical model. However, the equivalence theory that we propose is not complete, and it remains as future works to find a complete equivalence theory and to charac-

terize the necessary structures of the diagram for our categorical construction to work.

To summarize what we are going to introduce in this chapter, we propose a concrete, symmetric monoidal category M such that applying Rios&Selinger’s construction gives us access to an interpretation of dynamic lifting. The model we propose follows Moggi’s categorical interpretation of side effect [44] and models the action of measurement using a (strong) monad. Our semantics is therefore based on:

- A category of diagrams, serving as graph-like abstractions of quantum channels. This category is compact-closed and features products: it matches the requirements of the base category \overline{M} in Selinger&Rios’ work which interprets higher-order circuits. This category is discussed in Section 4.1.
- The category $\overline{\overline{M}}$, extending \overline{M} with the same procedure as Rios&Selinger. This category is the category of values, following Moggi’s computational interpretation. It is presented in Section 4.2.
- A strong monad on $\overline{\overline{M}}$ that we denote with F . This monad encapsulates computations involving measurements: a general term of Proto-Quipper-L is therefore interpreted inside the Kleisli category $\overline{\overline{M}}_F$: This is the main novelty compared to other models of Proto-Quipper-like languages [55, 24, 38], and the critical reason for the possibility to interpret dynamic lifting.

4.1 . Category of quantum channels (i.e., diagrams)

In this section, we aim to build a category of quantum channels. We first define a graph-based language: we call the corresponding terms diagrams to distinguish them from the terms of the quantum channel of Section 3.1.1. Diagrams are directed graphs composed of several types of nodes and edges labeled with marks. We then build the category \overline{M} out of these terms.

4.1.1 . Diagram

Marks

Formally, we define marks as in Definition 4.1.1.

Definition 4.1.1. Mark has the following grammar

$$\text{(Mark)} \quad M ::= I \mid q \mid M \otimes M \mid \boxplus_{i \in X} M_i \mid M^\perp$$

where X ranges over the class of sets. Marks are subject to the equivalence relation defined as the following rules:

- $\boxplus_{i \in I} \boxplus_{j \in J} M_{(i,j)} = \boxplus_{j \in J} \boxplus_{i \in I} M_{(i,j)}$
- $(M_1 \otimes M_2)^\perp = M_1^\perp \otimes M_2^\perp$
- $(\boxplus_{i \in I} M_i)^\perp = \boxplus_{i \in I} (M_i^\perp)$
- $(M^\perp)^\perp = M$
- $\boxplus_{l \in L} \boxplus_{x \in I} M_{(l,x)} = \boxplus_{x \in I} \boxplus_{l \in L} M_{(l,x)}$, where $L = [l_1, \dots, l_n]$
- $\boxplus_{x \in X} I = I$

Note that $\boxplus_{i \in \emptyset} M_i$ acts as a unit for \boxplus : we denote it with I . If $A = [A_1 \dots A_n]$ is a list of marks, we use the notation A^\otimes for $A_1 \otimes \dots \otimes A_n$. We also use a binary notation for \boxplus when the indexed set contains 2 elements: $\boxplus_{x \in \{a,b\}} A_x = A_a \boxplus A_b$.

Diagrams

A diagram is a (possibly infinite) directed graph with edges indexed with marks and built from elementary nodes and boxes. A diagram is not necessarily a connected graph. By graphical convention, all edges are flowing upward: a diagram is therefore acyclic.

Elementary nodes make the basic building blocks of diagrams: they are shown in Figure 4.1a. As we work with directed graphs, each edge connected to a node is either an input or an output for that node. There are several kinds of elementary nodes: the structural nodes for capturing the compact closed structure: \cup , \cap , \otimes , I and the swap-node (also written σ); the structural nodes for handling the product: \boxplus for the diagonal map and π for the projection; the structural nodes for pointing inputs in and outputs out of diagrams; the nodes specifically for quantum: (b) and (b) , with b ranging over booleans, where the former stands for initialization and the latter for projection onto the corresponding basis, (tr) for representing tracing (also useful for products), $\text{(G}_1\text{)}$ for unary unitary gates and $\text{(G}_2\text{)}$ for binary gates. As a graph, names over elementary nodes do not have meaning, but equivalence relations over the diagram, which will be presented later, may distinguish nodes with different names.

Note that the nodes allow more expressivity than what we need: for instance, (tr) and (b) are indistinguishable. We nonetheless keep them in order to draw attention to the correspondence with quantum computation and an obvious mapping to completely positive maps. For legibility, we do not draw in and out nodes unless necessary.

Presented in Figure 4.1b, a box-node is built from a family of diagrams. It envelopes a family of general diagrams indexed by a set I inside a box. They should all share the same input and output marks except for one pair of input/output (represented on the left of the box-node). As a node, box-node

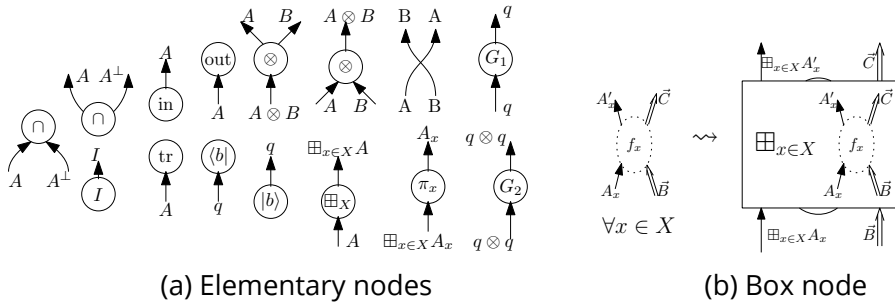


Figure 4.1: Diagram nodes

has the same input and output marks as its contained diagrams except that the left-most marks: these are the \boxplus of all left-most marks of the family. We represent the juxtaposition of edges as a double arrow. This node is the last piece needed for representing products.

Diagrams are inductively constructed by horizontal and vertical compositions of diagrams which is shown in Figure 4.2.

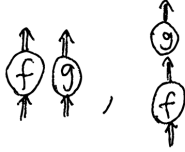


Figure 4.2: Horizontal and vertical composition of diagrams

Equivalence relation on diagrams

We define an equivalence relation on diagrams. The equivalence is given with local rules that can be extended to larger diagrams coherently: subgraphs can be rewritten inside a larger graph (which is a graph constructed by the structural rules of horizontal and vertical compositions and the box node constructor applied to subgraphs). These rules precisely capture what is needed for the categorical semantics to work. For instance, we include all of the rules for compact closed categories [61]. We also, for instance, need the fact that the (π) -node acts as a projection over box-nodes. The complete list is shown in Figure 4.3.

4.1.2 . Category of diagrams and its product

We define the category of diagrams from the definition of diagrams and show that this category is symmetric monoidal closed and is endowed with a product.

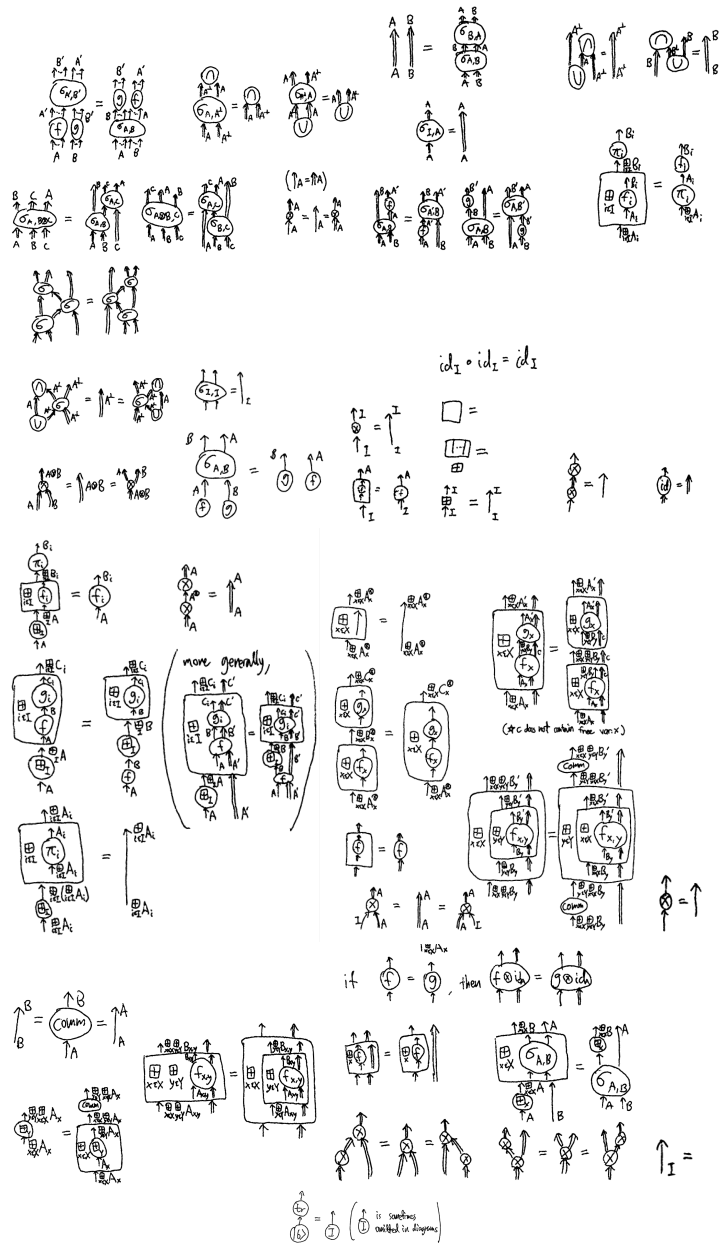


Figure 4.3: Equivalence relation of diagrams

Category of diagrams \overline{M}

Based on the definition of diagrams, we define the category of diagrams \overline{M} with the following data:

- objects are lists of marks $[A_1, \dots, A_n]$ and
- morphisms $[A_1, \dots, A_n] \rightarrow [B_1, \dots, B_m]$ are diagrams with $\textcircled{\text{in}}$ -nodes of marks A_i and $\textcircled{\text{out}}$ -nodes of marks B_i , modulo the equivalence relation

defined on diagrams.

We use the notation \vec{A} for the list of the A_i 's. An identity morphism is a diagram consisting of a bunch of simple edges connecting $\textcircled{\text{in}}$ and $\textcircled{\text{out}}$ nodes. Composition is defined by vertical composition of diagrams which consists in identifying $\textcircled{\text{out}}$ and $\textcircled{\text{in}}$ nodes of diagrams. Then we can show that the category satisfies associativity and unit law, hence \overline{M} is indeed a category .

The category \overline{M} is symmetric monoidal where we let the unit $I = []$ be the empty list, and let monoidal structure be given with the bi-functor $\otimes : \overline{M} \times \overline{M} \rightarrow \overline{M}$ defined as follows :

- for object $\vec{A} = [A_1, \dots, A_n]$ and $\vec{B} = [B_1, \dots, B_m]$,
 $\vec{A} \otimes \vec{B} = [A_1, \dots, A_n, B_1, \dots, B_m]$, and
- for morphisms $f : \vec{A} \rightarrow \vec{B}$ and $g : \vec{C} \rightarrow \vec{D}$, $f \otimes g : (\vec{A} \otimes \vec{C}) \rightarrow (\vec{B} \otimes \vec{D})$
 is the juxtaposition (horizontal composition) of diagrams f and g .

Lemma 4.1.1. *The tensor product \otimes is a functor from the cartesian category $\overline{M} \times \overline{M}$ to \overline{M} . Moreover, $(- \otimes \vec{B})$ is a functor from \overline{M} to \overline{M} which is defined as follows:*

- for object \vec{A} : $(- \otimes \vec{B})(\vec{A}) = \vec{A} \otimes \vec{B}$; and
- for morphism $f : \vec{A} \rightarrow \vec{A}'$: $(- \otimes \vec{B})(f) = f \otimes \text{id}_{\vec{B}}$.

Proof. The tensor product is a functor since it preserves the identity and composition:

- $\text{id}_{\vec{A}} \otimes \text{id}_{\vec{B}} = \text{id}_{\vec{A} \otimes \vec{B}}$ for $\vec{A}, \vec{B} \in \text{Obj}(\overline{M})$
- $(f' \circ f) \otimes (g' \circ g) = (f' \otimes g') \circ (f \otimes g)$ for the morphisms f and g , and f' and g'

Next, we can show that $(- \otimes \vec{B})$ is a functor by showing that it preserves the identity and composition as follows:

- $(- \otimes \vec{B})(\text{id}_{\vec{A}}) = \text{id}_{(- \otimes \vec{B})(\vec{A})} = \text{id}_{\vec{A} \otimes \vec{B}}$;
- $(- \otimes \vec{B})(g \circ f) = (g \circ f) \otimes \text{id}_{\vec{B}} = (g \otimes \text{id}_{\vec{B}}) \circ (f \otimes \text{id}_{\vec{B}}) = (- \otimes \vec{B})(g) \circ (- \otimes \vec{B})(f)$

□

As for the standard graphical representation of symmetric monoidal structure [61], the associativity, unit laws, and symmetry of the tensor product follow from their graphical conventions. In specific, associativity, unit law, and symmetry of the tensor product, are defined with the following natural transformations, which is morphism in \overline{M} assigned to each object in \overline{M} , $\overline{M} \times \overline{M}$, and $\overline{M} \times \overline{M} \times \overline{M}$:

- $\alpha_{\vec{A}, \vec{B}, \vec{C}} : (\vec{A} \otimes \vec{B}) \otimes \vec{C} \xrightarrow{\text{id}}_{\overline{M}} \vec{A} \otimes (\vec{B} \otimes \vec{C})$
- $l_{\vec{A}} : I \otimes \vec{A} \xrightarrow{\text{id}}_{\overline{M}} \vec{A}$ and $r_{\vec{A}} : \vec{A} \otimes I \xrightarrow{\text{id}}_{\overline{M}} \vec{A}$
Note that $l_{\vec{A}}$ and $r_{\vec{A}}$ relies on the fact that \otimes -node is equivalent to id according to the equivalence theory of diagrams (Figure 4.3).
- $\sigma_{\vec{A}, \vec{B}} : \vec{A} \otimes \vec{B} \rightarrow_{\overline{M}} \vec{B} \otimes \vec{A}$ is defined by using σ -node as shown in Figure 4.4.

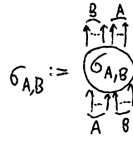


Figure 4.4: Definition of $\sigma_{\vec{A}, \vec{B}}$

Note that the naturality of the natural transformations $\alpha_{\vec{A}, \vec{B}, \vec{C}}$, $l_{\vec{A}}$, and $r_{\vec{A}}$ follows straightforwardly from the fact that they are identity morphism while the naturality of the symmetry $\sigma_{\vec{A}, \vec{B}}$ in Eq. (4.1) can be shown from the equation theory of diagram as shown in Figure 4.5.

$$\begin{array}{ccc}
 \vec{A} \otimes \vec{B} & \xrightarrow{\sigma_{\vec{A}, \vec{B}}} & \vec{B} \otimes \vec{A} \\
 \downarrow f \otimes g & & \downarrow g \otimes f \\
 \vec{A}' \otimes \vec{B}' & \xrightarrow{\sigma_{\vec{A}', \vec{B}'}} & \vec{B}' \otimes \vec{A}'
 \end{array} \tag{4.1}$$

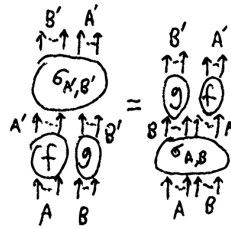


Figure 4.5: Proof of the naturality of symmetry of diagrams

Lemma 4.1.2 (Symmetric monoidal category \overline{M}). *The category of diagrams with tensor product $(\overline{M}, \otimes, I, \alpha, l, r, \sigma)$ as defined above is a symmetric monoidal category.*

Proof. It remains to show the coherence conditions.

Since the associativity α is identity the pentagon identity in Eq.(4.2) can be shown straightforwardly.

$$\begin{array}{ccc}
& (\vec{A} \otimes \vec{B}) \otimes (\vec{C} \otimes \vec{D}) & \\
\alpha_{\vec{A} \otimes \vec{B}, \vec{C}, \vec{D}} \nearrow & & \searrow \alpha_{\vec{A}, \vec{B}, \vec{C} \otimes \vec{D}} \\
((\vec{A} \otimes \vec{B}) \otimes \vec{C}) \otimes \vec{D} & & \vec{A} \otimes (\vec{B} \otimes (\vec{C} \otimes \vec{D})) \\
\alpha_{\vec{A}, \vec{B}, \vec{C}} \otimes \text{id}_{\vec{D}} \searrow & & \nearrow \text{id}_{\vec{A}} \otimes \alpha_{\vec{B}, \vec{C}, \vec{D}} \\
(\vec{A} \otimes (\vec{B} \otimes \vec{C})) \otimes \vec{D} & \xrightarrow{\alpha_{\vec{A}, \vec{B} \otimes \vec{C}, \vec{D}}} & \vec{A} \otimes ((\vec{B} \otimes \vec{C}) \otimes \vec{D})
\end{array} \tag{4.2}$$

Next, the hexagon identity in Eq. (4.3), follows from the equivalence of diagram shown in Figure 4.6.

$$\begin{array}{ccc}
(\vec{A} \otimes \vec{B}) \otimes \vec{C} & \xrightarrow{\alpha} & \vec{A} \otimes (\vec{B} \otimes \vec{C}) & \xrightarrow{\sigma} & (\vec{B} \otimes \vec{C}) \otimes \vec{A} \\
\downarrow \sigma \otimes \text{id} & & & & \downarrow \alpha \\
(\vec{B} \otimes \vec{A}) \otimes \vec{C} & \xrightarrow{\alpha} & \vec{B} \otimes (\vec{A} \otimes \vec{C}) & \xrightarrow{\text{id} \otimes \sigma} & \vec{B} \otimes (\vec{C} \otimes \vec{A})
\end{array} \tag{4.3}$$

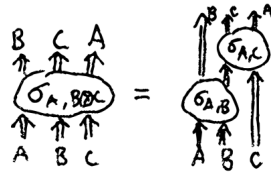


Figure 4.6: Proof of the coherence condition of associativity of the monoidal structure of \overline{M}

The following two commute diagrams in Eq.(4.4), which are proven in Figure 4.7, correspond to the involutiveness of the symmetry and the triangle identity, respectively.

$$\begin{array}{ccc}
\vec{A} \otimes \vec{B} & & I \otimes \vec{A} & \xrightarrow{\sigma} & \vec{A} \otimes I \\
\downarrow \sigma & \searrow \text{id} & \downarrow l & & \swarrow r \\
\vec{B} \otimes \vec{A} & \xrightarrow{\sigma} & \vec{A} \otimes \vec{B} & & \vec{A}
\end{array} \tag{4.4}$$

□

Internal hom in \overline{M}

The operation on marks $(-)^{\perp}$ lifts to a contravariant functor, giving a compact-closed structure to \overline{M} . It then admits an internal hom: $\vec{A} \multimap \vec{B}$ can be defined as $\vec{A}^{\perp} \multimap \vec{B}$ where \vec{A}^{\perp} refers to the list of the duals of marks of \vec{A} . Specifically,

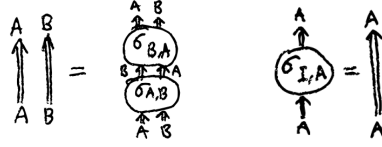


Figure 4.7: Proof of coherence conditions for the monoidal structure of \overline{M}

by letting $\vec{A} = [A_1, \dots, A_n]$ and $\vec{B} = [B_1, \dots, B_m]$, explicit representation of the internal hom can be obtained as in Eq. (4.5).

$$\begin{aligned}
 \vec{A} \multimap \vec{B} &= [A_1, \dots, A_n] \multimap [B_1, \dots, B_m] \\
 &= [A_1^\perp, \dots, A_n^\perp, B_1, \dots, B_m] \\
 &= [A_1^\perp, \dots, A_n^\perp] \otimes [B_1, \dots, B_m] \\
 &= \vec{A}^\perp \otimes \vec{B}
 \end{aligned} \tag{4.5}$$

This definition of internal hom relies on the interpretation of $(-)^{\perp}$ as the reversed edge. For example, a morphism from $[q]$ to $[q]$ in the category \overline{M} is a diagram with one incoming edge of mark q and one outgoing edge of mark q . This diagram is equivalently represented as two outgoing edges of marks q^{\perp} and q . Actually, this equivalence is what we need to show that \overline{M} is closed, namely, to show the isomorphism $\text{Hom}(\vec{A}, \vec{B} \multimap \vec{C}) \cong \text{Hom}(\vec{A} \otimes \vec{B}, \vec{C})$.

We now show this formally. First, note that $(- \otimes \vec{B})$ is a functor from Lemma 4.1.1. We can show that the functor has a right adjoint, $((- \otimes \vec{B}), (\vec{B} \multimap -), \eta, \epsilon)$ where the functor $(\vec{B} \multimap -)$ comes from the internal hom defined as follows:

- for object \vec{A} : $(\vec{B} \multimap -)(\vec{A}) = \vec{B}^\perp \otimes \vec{A}$;
- for morphism $f : \vec{A} \rightarrow \vec{A}'$: $(\vec{B} \multimap -)(f) = \text{id}_{\vec{B}^\perp} \otimes f$.

Again, note that we can show that $(\vec{B} \multimap -)$ is a functor.

Next, we define the natural transformations $\eta : \mathbf{1}_{\overline{M}} \rightarrow (\vec{B} \multimap -) \circ (- \otimes \vec{B})$ and $\epsilon : (- \otimes \vec{B}) \circ (\vec{B} \multimap -) \rightarrow \mathbf{1}_{\overline{M}}$. To begin with, let us examine the two compositions of functors $(\vec{B} \multimap -) \circ (- \otimes \vec{B})$ and $(- \otimes \vec{B}) \circ (\vec{B} \multimap -)$:

- $(\vec{B} \multimap -) \circ (- \otimes \vec{B})$:
 - for object \vec{A} : $(\vec{B} \multimap -) \circ (- \otimes \vec{B})(\vec{A}) = \vec{B}^\perp \otimes (\vec{A} \otimes \vec{B})$;
 - for morphism $f : \vec{A} \rightarrow \vec{A}'$: $(\vec{B} \multimap -) \circ (- \otimes \vec{B})(f) = \text{id}_{\vec{B}^\perp} \otimes (f \otimes \text{id}_{\vec{B}})$.
- $(- \otimes \vec{B}) \circ (\vec{B} \multimap -)$:

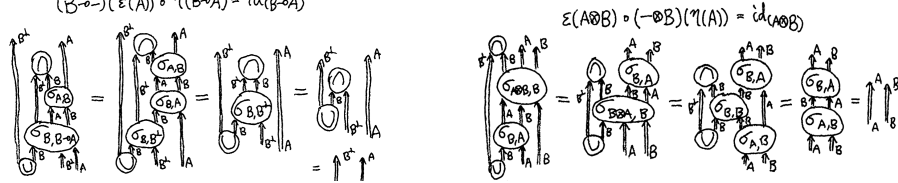
- for object $\vec{A} : (- \otimes \vec{B}) \circ (\vec{B} \multimap -)(\vec{A}) = (\vec{B}^\perp \otimes \vec{A}) \otimes \vec{B}$;
- for morphism $f : \vec{A} \rightarrow \vec{A}' : (- \otimes \vec{B}) \circ (\vec{B} \multimap -)(f) = (\text{id}_{\vec{B}^\perp} \otimes f) \otimes \text{id}_{\vec{B}}$.

We then define the natural transformations η and ϵ as in Table 4.1.

η	ϵ
for object $\vec{A}, \eta(\vec{A}) : \vec{A} \rightarrow (\vec{B} \multimap (\vec{A} \otimes \vec{B}))$ 	for object $\vec{A}, \epsilon(\vec{A}) : ((\vec{B} \multimap \vec{A}) \otimes \vec{B}) \rightarrow \vec{A}$
The naturality: $\begin{array}{ccc} \vec{A} & \xrightarrow{\eta(\vec{A})} & \vec{B} \multimap (\vec{A} \otimes \vec{B}) \\ \downarrow f & & \downarrow (\vec{B} \multimap -) \circ (- \otimes \vec{B})(f) \\ \vec{A}' & \xrightarrow{\eta(\vec{A}')} & \vec{B} \multimap (\vec{A}' \otimes \vec{B}) \end{array}$	The naturality: $\begin{array}{ccc} (\vec{B} \multimap \vec{A}) \otimes \vec{B} & \xrightarrow{\epsilon(\vec{A})} & \vec{A} \\ \downarrow (- \otimes \vec{B}) \circ (\vec{B} \multimap -)(f) & & \downarrow f \\ (\vec{B} \multimap \vec{A}') \otimes \vec{B} & \xrightarrow{\epsilon(\vec{A}')} & \vec{A}' \end{array}$

Table 4.1: Natural transformations η and ϵ for the internal hom in $\overline{\mathcal{M}}$

Then we can show that $((- \otimes \vec{B}), (\vec{B} \multimap -), \eta, \epsilon)$ is an adjunction as follows:
 $((\vec{B} \multimap -)\epsilon) \circ (\eta(\vec{B} \multimap -)) = \text{id}_{(\vec{B} \multimap -)}$ $(\epsilon(- \otimes \vec{B})) \circ ((- \otimes \vec{B})\eta) = \text{id}_{(- \otimes \vec{B})}$
 $(\vec{B} \multimap -)(\epsilon(A)) \circ \eta(\vec{B} \multimap A) = \text{id}_{(\vec{B} \multimap A)}$



As a corollary, we can show the isomorphism $\text{Hom}(\vec{A}, \vec{B} \multimap \vec{C}) \cong \text{Hom}(\vec{A} \otimes \vec{B}, \vec{C})$ in Lemma 4.1.3.

Lemma 4.1.3. For any object $\vec{A}, \vec{B}, \vec{C} \in \text{Obj}(\overline{\mathcal{M}})$, $\text{Hom}(\vec{A}, \vec{B} \multimap \vec{C}) \cong \text{Hom}(\vec{A} \otimes \vec{B}, \vec{C})$.

Proof. We define the following injections $\rightarrow_f : \text{Hom}(\vec{A}, \vec{B} \multimap \vec{C}) \rightarrow \text{Hom}(\vec{A} \otimes \vec{B}, \vec{C})$ and $\leftarrow_g : \text{Hom}(\vec{A} \otimes \vec{B}, \vec{C}) \rightarrow \text{Hom}(\vec{A}, \vec{B} \multimap \vec{C})$ and show

- that $\rightarrow_{\leftarrow_g} = g$, for any $g \in \text{Hom}(\vec{A} \otimes \vec{B}, \vec{C})$ and

- that $\leftarrow \rightarrow_f = f$, for any $f \in \mathbf{Hom}(\vec{A}, \vec{B} \multimap \vec{C})$.

First, we define the injections as in Eq. (4.6) which are represented explicitly as diagram in Figure 4.8.

$$\begin{aligned} \rightarrow_f &= \vec{A} \otimes \vec{B} \xrightarrow{(-\otimes \vec{B})(f)} (\vec{B} \multimap \vec{C}) \otimes \vec{B} \xrightarrow{\epsilon(\vec{C})} \vec{C} \\ \leftarrow_g &= \vec{A} \xrightarrow{\eta(\vec{A})} \vec{B} \multimap (\vec{C} \otimes \vec{B}) \xrightarrow{(\vec{B} \multimap -)(g)} \vec{B} \multimap \vec{C} \end{aligned} \tag{4.6}$$

Representation of the isomorphism over diagrams

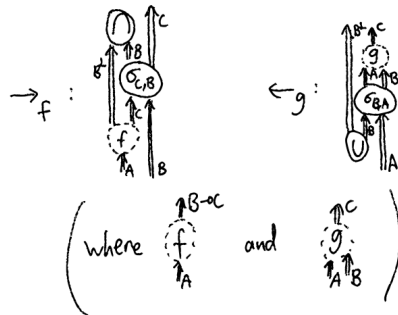


Figure 4.8: Representation of the isomorphisms as diagrams

Finally, the equality for the isomorphism can be shown as in Figure 4.9.

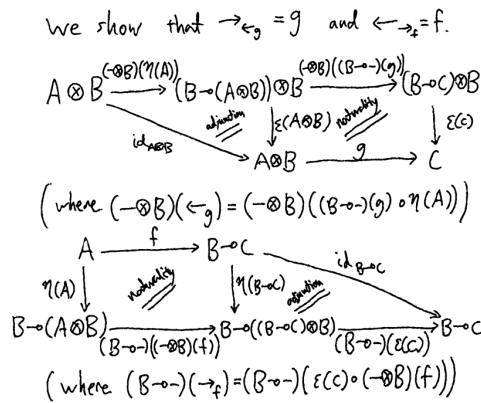


Figure 4.9: Proof of the isomorphism

□

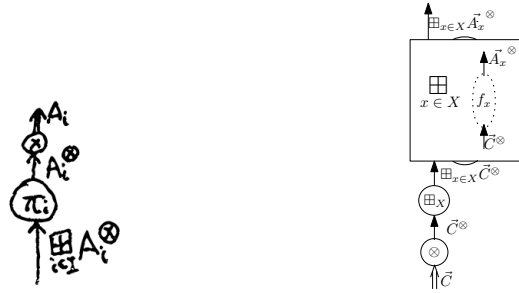
Product in \overline{M}

The category \overline{M} has products defined as follows. Thanks to the (π) -nodes and the corresponding diagram equivalence rules, product in the category \overline{M} can

be defined for any family of objects $\{\vec{A}_x \mid x \in X\}$ indexed by a set X by using \boxplus for mark as in Eq. (4.7).

$$\times_{x \in X} \vec{A}_x ::= \boxplus_{x \in X} \vec{A}_x = [\boxplus_{x \in X} \vec{A}_x^{\otimes}] \quad (4.7)$$

Moreover, the family of projections $\pi_x : (\times_{x \in X} \vec{A}_x) \rightarrow \vec{A}_x$ is given by the π -node as shown in Figure 4.10a. Finally, for any family of maps $\{f_x : \vec{C} \rightarrow \vec{A}_x\}_{x \in X}$, the morphisms $\langle f_x \rangle : \vec{C} \rightarrow (\times_{x \in X} \vec{A}_x)$ is the diagram presented in Figure 4.10b. As an abuse of notation, we use one \otimes for tensoring several wires at once.



(a) Projection π_i in \overline{M} (b) Pull back $\langle f_i \rangle$ of family $\{f_i\}$ in \overline{M}

Figure 4.10: Product in \overline{M}

To show formally that the category \overline{M} has a product, we need to show that $(\times_{x \in X} \vec{A}_x, (\pi_x)_{x \in X})$ is the pull back of the diagram of the tree with root and X leaves, i.e. for any $\vec{C} \in \text{Obj}(\overline{M})$ and family of morphisms $\{f_x : \vec{C} \rightarrow \vec{A}_x \mid x \in X\}$, there is a unique morphism $\langle f_x \rangle : \vec{C} \rightarrow (\times_{x \in X} \vec{A}_x)$ such that $f_x = \pi_x \circ \langle f_x \rangle$, for all $x \in X$. It then reduces to show the existence and the uniqueness of such morphism. By letting the pull back to be $\langle f_x \rangle$, the existence reduces to show that the morphism satisfies the following condition: for all $x \in X$, $f_x = \pi_x \circ \langle f_x \rangle$. Therefore, we can show the existence by using the equivalence theory of diagrams, as shown in Figure 4.11a.

Next, for the uniqueness (Eq. (4.8)), we use the fact that the following two propositions from Eq. 4.8 and Eq. 4.9 are equivalent given that $h = \langle f_x \rangle$ satisfies that $f_x = \pi_x \circ h$ for all $x \in X$ by the existence proof .

There is unique morphism $h : \vec{C} \rightarrow (\times_{x \in X} \vec{A}_x)$ such that $f_x = \pi_x \circ h$ for all $x \in X$. (4.8)

$$\text{For all } h : \vec{C} \rightarrow (\times_{x \in X} \vec{A}_x), \langle \pi_x \circ h \rangle = h. \quad (4.9)$$

The proof of Eq. 4.9 can then be found in Figure 4.11b.

Note that the universality in Eq. (4.9) of the product implies that for any $f : \vec{C} \rightarrow \times_{i \in I} A_i$,

$$\vec{C} \xrightarrow{f} \times_{x \in X} \vec{A}_x = \vec{C} \xrightarrow{\langle \pi_x \circ f \rangle} \times_{x \in X} \vec{A}_x \quad (4.10)$$

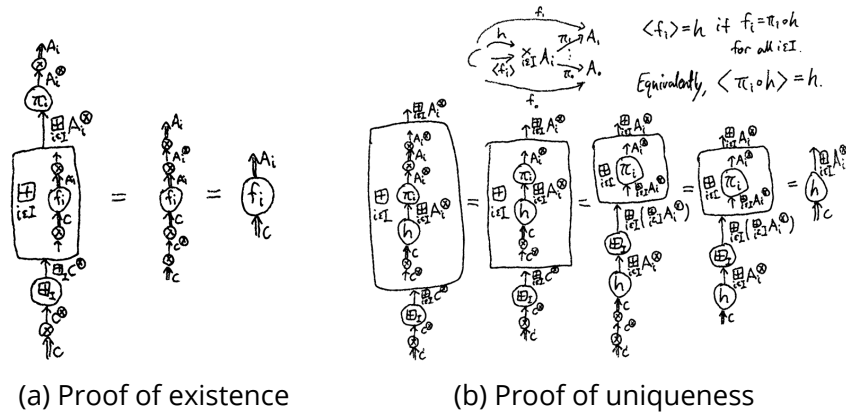


Figure 4.11: Proof of the proposition that $(\times_{x \in X} \vec{A}_x, (\pi_x)_{x \in X})$ is pull back

where $\times_{x \in X} \vec{A}_x = [\boxplus_{x \in X} A_x^\otimes]$. We can obtain the following corollary in Lemma 4.1.4 by letting $\vec{C} = I$.

Lemma 4.1.4. *For any $f : I \rightarrow_{\overline{M}} [\boxplus_{i \in I} A_i^\otimes]$, it is represented as:*

$$f = \langle \pi_i \circ f \rangle = \begin{array}{c} \uparrow \boxplus A_i^\otimes \\ \boxed{\pi_i} \\ \boxplus \\ \uparrow I \end{array}$$

Examples of morphisms in \overline{M}

We show in Figure 4.12 two interesting morphisms in the category \overline{M} . Note that morphisms in \overline{M} are equivalence classes of diagrams. First, the morphism $n : [q] \rightarrow [I \boxplus I]$ in Figure 4.12a corresponds to the measure. In each branch, we perform a projection, and we keep in the output the information of where we were. Note that the semantics does not state what is doing $\langle \# |$: what is essential is to (1) “remove” the q -wire, and (2) keep it as information if we are on the “true” or the “false” part.

Next, the morphism i in Figure 4.12b corresponds to qubit creation: it takes a boolean $I \boxplus I$, initializes a qubit depending on its state, and “forgets” the boolean.

As the last example, we can build the injections $I \rightarrow I \boxplus I$ in a similar way to n : first, a \boxplus -node, followed by a box-node where we trace out the component we do not need .

Based on these examples, we can interpret each quantum channel object from Section 3.1.1 as a morphism in \overline{M} as shown in Definition 4.1.2.

Definition 4.1.2 (Interpreting QCAlg terms). Let us use the notation $q^{\otimes n}$ to represent a list $[q, \dots, q]$ of size n . A QCAlg-term Q can be interpreted as a \overline{M} -morphism $\llbracket Q \rrbracket : A \rightarrow B$, where $A = q^{\otimes \text{in}(Q)}$ and B of tree-shape of the form



(a) Diagram for measurement (b) Diagram for qubit initialization

Figure 4.12: Examples of morphisms in \overline{M}

$\boxplus_{x \in X} (q^{\otimes n_x})$, following the tree-shape of $\vec{\text{out}}(Q)$. The \overline{M} -morphism $\llbracket Q \rrbracket$ is then defined by induction, using the idea presented above: initialization and unitary gates are simply composed, and the branches of a meas operation are encapsulated inside box-nodes.

4.1.3 . Compact closed category of diagrams

Finally, we show that we can make a compact closed category out of the category of diagrams \overline{M} . Recall that a compact closed category is a symmetric monoidal category where each object \vec{A} has its dual object \vec{A}^\perp together with a unit $\eta_A : I \rightarrow \vec{A}^\perp \otimes \vec{A}$ and a counit $\epsilon_A : \vec{A} \otimes \vec{A}^\perp \rightarrow I$ maps which satisfies the following conditions [60]:

$$l_{\vec{A}} \circ (\epsilon_{\vec{A}} \otimes \text{id}_{\vec{A}}) \circ \alpha_{\vec{A}, \vec{A}^\perp, \vec{A}}^{-1} \circ (\text{id}_{\vec{A}} \otimes \eta_{\vec{A}}) \circ r_{\vec{A}}^{-1} = \text{id}_{\vec{A}} \quad (4.11)$$

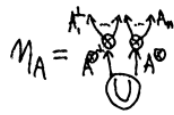
$$r_{\vec{A}^\perp} \circ (\text{id}_{\vec{A}^\perp} \otimes \epsilon_{\vec{A}}) \circ \alpha_{\vec{A}^\perp, \vec{A}, \vec{A}^\perp} \circ (\eta_{\vec{A}} \otimes \text{id}_{\vec{A}^\perp}) \circ l_{\vec{A}^\perp}^{-1} = \text{id}_{\vec{A}^\perp} \quad (4.12)$$

Concretely, we define the dual as in Definition 4.1.3.

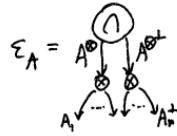
Definition 4.1.3 (Dual in the compact closed category of diagrams).

$$A^\perp = [A_1, \dots, A_n]^\perp = [A_1^\perp, \dots, A_n^\perp].$$

To show that it is a dual, we define the morphisms $\eta_A : I \rightarrow A^\perp \otimes A$ and $\epsilon_A : A \otimes A^\perp \rightarrow I$ as in Definition 4.1.4.



(a) Unit map



(b) Counit map

Figure 4.13: Unit and counit maps in the compact closed category of diagram

Definition 4.1.4 (Unit and counit maps in the compact closed category).

Then, the triangle identities from Eq. (4.11) and Eq. (4.12) can be shown as in Table 4.2. Note that the equations are simplified by using the fact that $l_{\vec{A}} = \text{id}_{\vec{A}} = r_{\vec{A}}$.

$$\begin{array}{ccc}
 \vec{A} \otimes I & \xrightarrow{\text{id} \otimes \eta_{\vec{A}}} & \vec{A} \otimes (\vec{A}^\perp \otimes \vec{A}) & & I \otimes \vec{A}^\perp & \xrightarrow{\eta_{\vec{A}} \otimes \text{id}} & (\vec{A}^\perp \otimes \vec{A}) \otimes \vec{A}^\perp \\
 \downarrow & & \downarrow \alpha^{-1} & & \downarrow & & \downarrow \alpha \\
 I \otimes \vec{A} & \xleftarrow{\epsilon_{\vec{A}} \otimes \text{id}} & (\vec{A} \otimes \vec{A}^\perp) \otimes \vec{A} & & \vec{A}^\perp \otimes I & \xleftarrow{\text{id} \otimes \eta_{\vec{A}}} & \vec{A}^\perp \otimes (\vec{A} \otimes \vec{A}^\perp)
 \end{array}$$

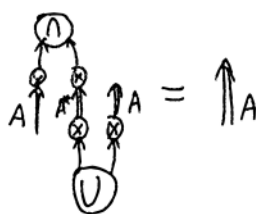
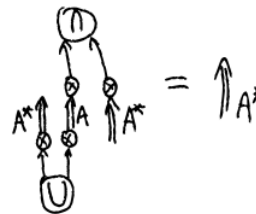



Table 4.2: Proof of the triangle identities for compact closed structure

4.2 . Rios and Selinger completion

Coproduct completion allows us to define families of circuits [55, 24]. The categorical structure clearly separates what is purely quantum and what is parameter to the computation, and we have parametric families of quantum channels.

Formally, this is done using the coproduct completion $\overline{\overline{M}}$ of \overline{M} . In this completion, an object corresponds to a pair $(X, (A_x)_{x \in X})$ where X is a set and A_x is an object of \overline{M} for each $x \in X$: this should be understood as a parametric families of objects of \overline{M} . A morphism from $(X, (A_x))$ to $(Y, (B_y))$ corresponds to a pair $(f_0, (f_x)_{x \in X})$ where $f_0 : X \rightarrow Y$ is a set function and $f_x : A_x \rightarrow B_{f_0(x)}$ is a morphism in \overline{M} for each $x \in X$. Intuitively, to each choice of parameter x we have a \overline{M} -morphisms $A_x \rightarrow A_{f_0(x)}$.

Composition is defined with $(g_0, (g_y)) \circ (f_0, (f_x)) = (g_0 \circ f_0, (g_{f_0(x)} \circ f_x))$ where $(g_0, (g_y)) : (Y, (B_y)) \rightarrow (Z, (C_z))$ and $(f_0, (f_x)) : (X, (A_x)) \rightarrow (Y, (B_y))$ are morphisms in $\overline{\overline{M}}$, while the identity is $\text{id}_A = (\text{id}_X, (\text{id}_{A_x}))$ for an object $A = (X, (A_x))$.

According to Rios&Selinger [55], the category $\overline{\overline{M}}$ is symmetric monoidal closed and features products and coproducts. In particular, the monoidal unit is $(\{\emptyset\}, (I))$ (where \emptyset stands for the only representative of the singleton-set), and when $A = (X, (A_x))$ and $B = (Y, (B_y))$, the tensor on objects is $A \otimes B = (X \times Y, (A_x \otimes B_y)_{(x,y)})$ and the internal hom is $A \multimap B = (X \rightarrow$

$Y, (C_f)_{f \in X \rightarrow Y}$ ($X \rightarrow Y$ is the set of all set-functions from X to Y and C_f refers to the product $\boxplus_{x \in X} (A_x \multimap B_{f(x)})$ of internal homs in \overline{M}). Note that the product is defined by \boxplus in the case of the category of diagrams. Also, note that compared to [55], we can capitalize on the concrete structure of the category for the proofs involving the coproduct completion. For instance, associativity is trivial in our category \overline{M} .

Finally, in order to model the type operator “!” , Rios&Selinger rely on Benton’s linear/non-linear model [12], based on an adjunction between a symmetric monoidal closed category and a cartesian closed category. In our case, as in [55] the adjunction is built between the SMCC \overline{M} and the cartesian closed category **Set**. The two functors of the adjunction are $p : \mathbf{Set} \rightarrow \overline{M}$, defined on objects as $p(X) = (X, (I)_X)$, and $b : \overline{M} \rightarrow \mathbf{Set}$, defined on objects as $b(X, (A_x)) = \sum_{x \in X} \overline{M}(I, A_x)$ where $\overline{M}(I, A_x)$ is the set of morphisms between the objects I and A_x of the category \overline{M} and $\sum_{x \in X} \overline{M}(I, A_x)$ is the disjoint union of all such sets over X . From the adjunction, one can then construct a comonad “!” defined as $! = p \circ b$ (see the appendix for more details).

Remark. In \overline{M} there are two classes of interesting objects. The *parameters* are objects of the form $(X, (I)_{x \in X})$: the family consists of trivial objects of \overline{M} , and the only information is given by...the parameter. The *state* object is the dual: the parameter is trivial, and the family is of size 1 with only one object of \overline{M} . It is then of the form $(\{\emptyset\}, (A))$. One therefore has two booleans: a parameter boolean $b_p = (\{\#\}, \{\#\}, (I)_{\{\#\}, \{\#\}})$, and the state boolean $b_s = (\{\emptyset\}, (I \boxplus I))$ living in \overline{M} .

4.2.1 . Definition of coproduct completion

Coproduct completion in Definition 4.2.1 allows us to define a family of circuits [55, 24].

Definition 4.2.1 (Coproduct completion of a symmetric monoidal closed category). The coproduct completion of a symmetric monoidal category \overline{M} is defined with the following data.

- objects correspond to a pair $(X, (A_x)_{x \in X})$ where X is a set and A_x is an object of \overline{M} for each $x \in X$;
- morphisms from $(X, (A_x))$ to $(Y, (B_y))$ correspond to a pair $(f_0, (f_x)_{x \in X})$ where $f_0 : X \rightarrow Y$ is a set function and $f_x : A_x \rightarrow B_{f_0(x)}$ is a morphism in \overline{M} for each $x \in X$.

We call the coproduct completion of \overline{M} as $\overline{\overline{M}}$ following the convention in [55]. The composition and identity morphism are defined as follows:

- composition : $(g_0, (g_y)) \circ (f_0, (f_x)) = (g_0 \circ f_0, (g_{f_0(x)} \circ f_x))$ where $(g_0, (g_y)) : (Y, (B_y)) \rightarrow (Z, (C_x))$ and $(f_0, (f_x)) : (X, (A_x)) \rightarrow (Y, (B_y))$ are morphisms in $\overline{\overline{M}}$;

- identity : $\text{id}_A = (\text{id}_X, (\text{id}_{A_x}))$ for an object $A = (X, (A_x))$.

Next, we show that the above construction in Definition 4.2.1 produces a category. It suffices to check associativity and unit laws to ensure that $\overline{\overline{M}}$ forms a valid category. The following proofs show associativity and unit laws in $\overline{\overline{M}}$.

- associativity law : suppose $(X, (A_x)) \xrightarrow{f} (Y, (B_y)) \xrightarrow{g} (Z, (C_z)) \xrightarrow{h} (W, (D_w))$, then

$$\begin{aligned} h \circ (g \circ f) &= h \circ (g_0 \circ f_0, (g_{f_0(x)} \circ f_x)) \\ &= (h_0 \circ g_0 \circ f_0, (h_{(g_0 \circ f_0)(x)} \circ g_{f_0(x)} \circ f_x)) \\ &= (h_0 \circ g_0, (h_{g_0(x)} \circ g_x)) \circ f \\ &= (h \circ g) \circ f \end{aligned}$$

from the associativity of the category **Set** and \overline{M} .

- unit law : for a morphism $f : A \rightarrow B$ where $A = (X, (A_x))$ and $B = (Y, (B_y))$,

$$\begin{aligned} f \circ \text{id}_A &= (f_0 \circ \text{id}_X, (f_x \circ \text{id}_{A_x})) \\ &= (f_0, (f_x)) \\ \text{id}_B \circ f &= (\text{id}_Y \circ f_0, (\text{id}_{B_{f_0(x)}} \circ f_x)) \end{aligned}$$

where $f_0 \circ \text{id}_X = f_0 = \text{id}_Y \circ f_0$ follows from the unit law of **Set** and $f_x \circ \text{id}_{A_x} = f_x = \text{id}_{B_{f_0(x)}} \circ f_x$ follows from the unit law of \overline{M} for each $x \in X$.

4.2.2 . Symmetric monoidal category $\overline{\overline{M}}$ from coproduct completion

According to Rios and Selinger, unit (I), tensor product (\otimes), and internal hom (\multimap) for the symmetric monoidal closed structure of $\overline{\overline{M}}$ is defined as in Definition 4.2.2.

Definition 4.2.2. The category $\overline{\overline{M}}$ obtained by coproduct completion of a symmetric monoidal closed category \overline{M} is symmetric monoidal closed with the following structure:

- $I = (\{\emptyset\}, (I))$;
- $A \otimes B = (X \times Y, (A_x \otimes B_y)_{(x,y)})$ where $A = (X, (A_x))$ and $B = (Y, (B_y))$;
- $A \multimap B = (X \rightarrow Y, (C_f)_{f \in X \rightarrow Y})$ where $X \rightarrow Y$ is the set of functions from X to Y and C_f refers to the product of internal homs in \overline{M} , $\prod_{x \in X} (A_x \multimap B_{f(x)})$, for each function $f : X \rightarrow Y$. Note that the product is defined by \boxplus in the case of the category of diagrams, hence, we explicitly write the internal hom as follows:

$$(X, (A_x)) \multimap (Y, (B_y)) = (X \rightarrow Y, (\boxplus_{x \in X} (A_x \multimap B_{f(x)}))_{f \in X \rightarrow Y})$$

where $A_x \multimap B_{f(x)}$ is the internal hom in \overline{M} .

On top of it, we capitalize on the concrete structure of the category to simplify the proofs involving the coproduct completion. To explain it, let us look in detail at the symmetric monoidal closed structure of $\overline{\overline{M}}$.

Symmetric monoidal structure

First, we focus on the symmetric monoidal structure of $\overline{\overline{M}}$. The structure comes with natural transformations $\alpha_{A,B,C} : (A \otimes B) \otimes C \rightarrow_{\overline{\overline{M}}} A \otimes (B \otimes C)$, $l_A : I \otimes A \rightarrow_{\overline{\overline{M}}} A$, $r_A : A \otimes I \rightarrow_{\overline{\overline{M}}} A$, and $\sigma_{A,B} : A \otimes B \rightarrow_{\overline{\overline{M}}} B \otimes A$ which are defined as in Eq. (4.13).

$$\begin{aligned}
\alpha_{A,B,C} &::= (\{(x, y), z\} \mapsto (x, (y, z))), (\alpha_{A_x, B_y, C_z})_{((x, y), z)} \\
&= (\{(x, y), z\} \mapsto (x, (y, z))), (\text{id}_{(A_x \otimes B_y) \otimes C_z}) \\
l_A &::= (\{\emptyset, x\} \mapsto x), (l_{A_x}) \\
&= (\{\emptyset, x\} \mapsto x), (\text{id}_{A_x}) \\
r_A &::= (\{x, \emptyset\} \mapsto x), (r_{A_x}) \\
&= (\{x, \emptyset\} \mapsto x), (\text{id}_{A_x}) \\
\sigma_{A,B} &::= (\{x, y\} \mapsto (y, x)), (\sigma_{A_x, B_y})
\end{aligned} \tag{4.13}$$

Note that, for simplicity, we identify $\alpha_{A,B,C}$, l_A , and r_A be identity morphism in $\overline{\overline{M}}$. It means that we equate the set $\{(x, y), z\}$ and $\{x, (y, z)\}$; $\{\emptyset, x\}$ and $\{x\}$; and $\{x, \emptyset\}$ and $\{x\}$. Moreover, we let α_{A_x, B_y, C_z} , l_{A_x} , and r_{A_x} be identity morphisms in the category of diagrams $\overline{\overline{M}}$. As consequence, all coherence conditions for the symmetric monoidal structure become trivial except the coherence conditions related to symmetry shown in Eq. (4.14), Eq. (4.2.2), and Eq. (4.16).

$$\begin{array}{ccc}
(A \otimes B) \otimes C & \xrightarrow{\alpha} & A \otimes (B \otimes C) & \xrightarrow{\sigma} & (B \otimes C) \otimes A \\
\downarrow \sigma \otimes \text{id} & & & & \downarrow \alpha \\
(B \otimes A) \otimes C & \xrightarrow{\alpha} & B \otimes (A \otimes C) & \xrightarrow{\text{id} \otimes \sigma} & B \otimes (C \otimes A)
\end{array} \tag{4.14}$$

$$\begin{array}{ccc}
A \otimes B & & \\
\downarrow \sigma & \searrow \text{id} & \\
B \otimes A & \xrightarrow{\sigma} & A \otimes B
\end{array} \tag{4.15}$$

$$\begin{array}{ccc}
I \otimes A & \xrightarrow{\sigma} & A \otimes I \\
\searrow l & & \swarrow r \\
& & A
\end{array} \tag{4.16}$$

These properties can be shown for the definition of $\overline{\overline{M}}$ as follows:

- Eq. (4.14) follows from the coherence rule of SMCC \overline{M} .

$$\begin{aligned}
& (A \otimes B) \otimes C \xrightarrow{\alpha} A \otimes (B \otimes C) \xrightarrow{\sigma} (B \otimes C) \otimes A \xrightarrow{\alpha} B \otimes (C \otimes A) \\
& = (\{((x, y), z) \mapsto (y, (z, x))\}, (\alpha_{B_y, C_z, A_x} \circ \sigma_{A_x, B_y \otimes C_z} \circ \alpha_{A_x, B_y, C_z})) \\
& = (\{((x, y), z) \mapsto (y, (z, x))\}, ((\text{id}_{B_y} \otimes \sigma_{A_x, C_z}) \circ \alpha_{B_y, A_x, C_z} \circ (\sigma_{A_x, B_y} \otimes \text{id}_{C_z}))) \\
& = (A \otimes B) \otimes C \xrightarrow{\sigma \otimes \text{id}} (B \otimes A) \otimes C \xrightarrow{\alpha} B \otimes (A \otimes C) \xrightarrow{\text{id} \otimes \sigma} B \otimes (C \otimes A)
\end{aligned}$$

- Eq. (4.2.2) can be shown as follows.

$$\begin{aligned}
A \otimes B & \xrightarrow{\sigma} B \otimes A \xrightarrow{\sigma} A \otimes B = (\{(x, y) \mapsto (x, y)\}, (\sigma_{B_y, A_x} \circ \sigma_{A_x, B_y})) \\
& = (\{(x, y) \mapsto (x, y)\}, (\text{id}_{A_x \otimes B_y})) \\
& = A \otimes B \xrightarrow{\text{id}} A \otimes B
\end{aligned}$$

- Eq. (4.16) follows from the coherence conditions in \overline{M} .

$$\begin{aligned}
I \otimes A & \xrightarrow{\sigma} A \otimes I \xrightarrow{r} A = (\{(\emptyset, x) \mapsto x\}, (r_{A_x} \circ \sigma_{I, A_x})) \\
& = (\{(\emptyset, x) \mapsto x\}, (l_{A_x})) \\
& = I \otimes A \xrightarrow{l} A
\end{aligned}$$

Internal hom in \overline{M}

Although Definition 4.2.2 gives us the definition of internal hom, we need to prove that our concrete category admits the symmetric monoidal closed structure. It can be shown by the following concrete definition of the adjunction $((-\otimes B), (B \multimap -), \eta : \mathbf{1}_{\overline{M}} \rightarrow (B \multimap -) \circ (-\otimes B), \epsilon : (-\otimes B) \circ (B \multimap -) \rightarrow \mathbf{1}_{\overline{M}})$ for the internal hom in \overline{M} .

To begin with, we define the functors $(-\otimes B)$ and $(B \multimap -)$ as in Table 4.3.

Next, we define the natural transformations η and ϵ as in Table 4.4.

Lemma 4.2.1. *The definition above admits that $((-\otimes B), (B \multimap -), \eta, \epsilon)$ is an adjunction, i.e. the following conditions are proved:*

$$\begin{aligned}
((B \multimap -)\epsilon) \circ (\eta(B \multimap -)) &= \text{id}_{B \multimap -} \\
(\epsilon(-\otimes B)) \circ ((-\otimes B)\eta) &= \text{id}_{-\otimes B}
\end{aligned}$$

which are represented in the following commute diagrams: for any object A ,

$$\begin{array}{ccc}
B \multimap A & & A \otimes B \\
\eta(B \multimap A) \downarrow & \searrow \text{id}_{B \multimap A} & \downarrow (-\otimes B)(\eta(A)) \\
B \multimap ((B \multimap A) \otimes B) & \xrightarrow{(\text{id}_{B \multimap -})(\epsilon(A))} & B \multimap A & \quad & (B \multimap (A \otimes B)) \otimes B & \xrightarrow{\epsilon(A \otimes B)} & A \otimes B \\
& & & & \text{id}_{A \otimes B} \searrow & &
\end{array}$$

Intuitively, we consider the morphism $\epsilon(A) : (B \multimap A) \otimes B \rightarrow_{\overline{M}} A$ as the evaluation and give the name $\text{eval}_{B, A}$ to the morphism.

<p>functor $(- \otimes B) : \overline{\overline{M}} \rightarrow \overline{\overline{M}}$</p> <p>for object $A = (X, (A_x))$,</p> $(- \otimes B)(A) = (X \times Y, (A_x \otimes B_y)_{(x,y)})$	<p>functor $(B \multimap -) : \overline{\overline{M}} \rightarrow \overline{\overline{M}}$</p> <p>for object $A = (X, (A_x))$,</p> $(B \multimap -)(A) = (Y \rightarrow X, ([\boxplus_{y \in Y} (B_y \multimap A_{f(y)})]^\otimes)_{f \in Y \rightarrow X})$
<p>for morphism $f = (f_0, (f_x)) : A \rightarrow_{\overline{\overline{M}}} A'$,</p> $(- \otimes B)(f) = (\{(x, y) \mapsto (f_0(x), y)\}, (f_x, \otimes \mathbf{id}_{B_y})_{(x,y)})$	<p>for morphism $f = (f_0, (f_x)) : A \rightarrow_{\overline{\overline{M}}} A'$,</p> $(B \multimap -)(f) = (\{p \mapsto f_0 \circ p \mid p : Y \rightarrow X\},$ <div style="text-align: center;"> <p style="text-align: right;">$\}_{p \in Y \rightarrow X})$</p> </div>
<p>The following axioms follow from the definition.</p> $(- \otimes B)(\mathbf{id}_A) = \mathbf{id}_{A \otimes B}$ $(- \otimes B)(g \circ f) = (- \otimes B)(g) \circ (- \otimes B)(f)$	<p>The following axioms follow from the definition.</p> $(B \multimap -)(\mathbf{id}_A) = \mathbf{id}_{B \multimap A}$ $(B \multimap -)(g \circ f) = (B \multimap -)(g) \circ (B \multimap -)(f)$

Table 4.3: Definition of functors $(- \otimes B)$ and $(B \multimap -)$ in $\overline{\overline{M}}$

4.2.3 . Finite coproduct in $\overline{\overline{M}}$

Another essential property of $\overline{\overline{M}}$ is the finite coproduct. Explicitly, for any $A, B \in \text{Obj}(\overline{\overline{M}})$, there exists an object $A + B$, i.e. coproduct of A and B , and morphisms

$$i_1 : A \rightarrow A + B, \text{ and } i_2 : B \rightarrow A + B$$

such that for any object C and morphisms $f : A \rightarrow C$ and $g : B \rightarrow C$, there exists unique $(f, g) : A + B \rightarrow C$ which satisfies that

$$f = (f, g) \circ i_1, \text{ and } g = (f, g) \circ i_2.$$

In the coproduct completion, the coproduct is defined as following bifunctor $+$: $\overline{\overline{M}} \times \overline{\overline{M}} \rightarrow \overline{\overline{M}}$:

- for an object $(A, B) \in \text{Obj}(\overline{\overline{M}} \times \overline{\overline{M}})$: $A + B = (X + Y, (A_x) :: (B_y))$;
- for a morphism $(f : A \rightarrow A', g : B \rightarrow B') \in \text{Hom}_{\overline{\overline{M}} \times \overline{\overline{M}}}((A, B), (A', B'))$:

$$f + g = (\{(0, x) \mapsto (0, f_0(x)), (1, y) \mapsto (1, g_0(y))\}, (f_x)_X :: (g_y)_Y) : A + B \rightarrow A' + B'$$

the functor:

- identity:

$$\text{for } (\text{id}_A, \text{id}_B) \in \text{Hom}_{\overline{\overline{M \times M}}}((A, B), (A, B)),$$

$$\begin{aligned} \text{id}_A + \text{id}_B &= (\{(0, x) \mapsto (0, x), (1, y) \mapsto (1, y)\}, (\text{id}_{A_x})_X \text{ :: } (\text{id}_{B_y})_Y) \\ &= (\text{id}_{X+Y}, (\text{id}_{A_x})_X \text{ :: } (\text{id}_{B_y})_Y) \\ &= \text{id}_{A+B} \end{aligned}$$

- composition:

$$\text{for } (f_1, f_2) \in \text{Hom}_{\overline{\overline{M \times M}}}((A, B), (A', B')) \text{ and } (g_1, g_2) \in \text{Hom}_{\overline{\overline{M \times M}}}((A', B'), (A'', B'')),$$

$$\begin{aligned} (g_1 + g_2) \circ (f_1 + f_2) &= (\{(0, x') \mapsto (0, g_1(x')), (1, y') \mapsto (1, g_2(y'))\}, (g_{1_{x'}})_{X'} \text{ :: } (g_{2_{y'}})_{Y'}) \\ &\quad \circ (\{(0, x) \mapsto (0, f_1(x)), (1, y) \mapsto (1, f_2(y))\}, (f_{1_x})_X \text{ :: } (f_{2_y})_Y) \\ &= (\{(0, x) \mapsto (0, g_1(f_1(x))), (1, y) \mapsto (1, g_2(f_2(y)))\}, \\ &\quad (g_{1_{f_1(x)}} \circ f_{1_x})_X \text{ :: } (g_{2_{f_2(y)}} \circ f_{2_y})_Y) \\ &= (g_1 \circ f_1) + (g_2 \circ f_2) \end{aligned}$$

It is equipped with the following natural transformations i_1 and i_2 :

- $i_1(A, B) = (\{x \mapsto (0, x)\}, (\text{id}_{A_x})) : A \rightarrow A + B$ which satisfies the following naturality diagram:

$$\begin{array}{ccc} A & \xrightarrow{i_1(A, B)} & A + B \\ \downarrow f & & \downarrow f+g \\ A' & \xrightarrow{i_1(A', B')} & A' + B' \end{array}$$

where

$$\begin{aligned} (f + g) \circ i_1(A, B) &= (\{x \mapsto (0, f_0(x))\}, (f_x)_X) \\ &= i_1(A', B') \circ f \end{aligned}$$

- $i_2(A, B) = (\{y \mapsto (1, y)\}, (\text{id}_{B_y})) : B \rightarrow A + B$ which is subject to the following naturality diagram:

$$\begin{array}{ccc} B & \xrightarrow{i_2(A, B)} & A + B \\ \downarrow g & & \downarrow f+g \\ B' & \xrightarrow{i_2(A', B')} & A' + B' \end{array}$$

where

$$\begin{aligned} (f + g) \circ i_2(A, B) &= (\{y \mapsto (1, g_0(y))\}, (g_y)_Y) \\ &= i_2(A', B') \circ g \end{aligned}$$

Given the definition, we can show the universality of the coproduct. For any morphisms $f : A \rightarrow C$ and $g : B \rightarrow C$, let $(f, g) = (\{(0, x) \mapsto f_0(x), (1, y) \mapsto g_0(y)\}, (f_x)_{(0,x)} \mathbin{::} (g_y)_{(1,y)}) : A + B \rightarrow C$. Then, we can check that

$$\begin{aligned}(f, g) \circ i_1 &= (\{x \mapsto (0, x) \mapsto f_0(x)\}, (f_x \circ \text{id}_{A_x})_X) = f \\ (f, g) \circ i_2 &= (\{y \mapsto (1, y) \mapsto g_0(y)\}, (g_y \circ \text{id}_{B_y})_Y) = g\end{aligned}$$

Moreover, there is unique morphism $(f, g) = A + B \rightarrow C$ which satisfies above conditions. Suppose that a morphism $h : A + B \rightarrow C$ satisfies that $f = h \circ i_1$ and $g = h \circ i_2$. Without loss of generality, let $h = (\{(0, x) \mapsto p(x), (1, y) \mapsto q(y)\}, (h_x)_{(0,x)} \mathbin{::} (h_y)_{(1,y)})$. Then since $f = h \circ i_1$, $p(x) = f(x)$ and $h_x = f_x$; and since $g = h \circ i_2$, $q(y) = g(y)$ and $h_y = g_y$.

Lastly, Lemma 4.2.2 shows that we can decompose morphisms by the universality.

Lemma 4.2.2. For $(f_1, f_2) \in \text{Hom}_{\overline{\overline{M \times M}}}(A_1, A_2), (B_1, B_2)$ and $(g_1, g_2) \in \text{Hom}_{\overline{\overline{M \times M}}}(B_1, B_2), (C, C)$, the following equation holds:

$$(g_1 \circ f_1, g_2 \circ f_2) = (g_1, g_2) \circ (f_1 + f_2).$$

Proof. Proof is in Figure 4.14 □

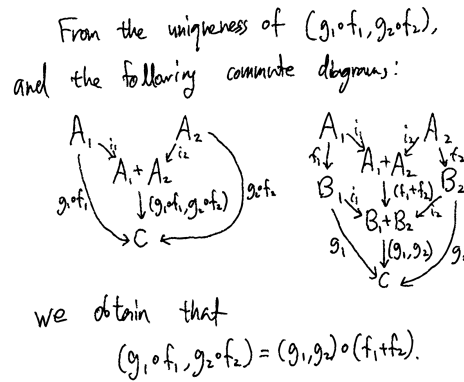


Figure 4.14: Proof of the commutativity of the coproduct and the composition

4.2.4 . Distributivity of \otimes over $+$

There is a natural transformation $\star : (- + -) \otimes C \rightarrow (- \otimes C) + (- \otimes C)$ for each object $C \in \text{Obj}(\overline{\overline{M}})$ defined as follows:

$$\begin{aligned}\star(A, B) &= (\{((0, x), z) \mapsto (0, (x, z)), ((1, y), z) \mapsto (1, (y, z))\}, \\ &\quad (\text{id}_{A_x \otimes C_z})_{((0,x),z)} \mathbin{::} (\text{id}_{B_y \otimes C_z})_{((1,y),z)}) \\ &: (A + B) \otimes C \rightarrow_{\overline{\overline{M}}} (A \otimes C) + (B \otimes C).\end{aligned}$$

It satisfies the naturality in Eq. (4.17) which can be shown as in Figure 4.15.

$$\begin{array}{ccc}
 (A + B) \otimes C & \xrightarrow{\star} & A \otimes C + B \otimes C \\
 \downarrow (f+g) \otimes h & & \downarrow f \otimes h + g \otimes h \\
 (A' + B') \otimes C' & \xrightarrow{\star} & A' \otimes C' + B' \otimes C'
 \end{array} \quad (4.17)$$

$$\begin{aligned}
 (f \otimes h + g \otimes h) \circ \star &= \left\{ \left((0, x), z \right) \mapsto (0, (x, z)) \mapsto (0, (f_0(x), h_0(z))), \right. \\
 &\quad \left. ((1, y), z) \mapsto (1, (y, z)) \mapsto (1, (g_0(y), h_0(z))) \right\}, \\
 &\quad \left(\begin{array}{c} f_x \\ \oplus \\ h_x \end{array} \right)_{(0,x,z)} \quad \text{::} \quad \left(\begin{array}{c} f_y \\ \oplus \\ h_y \end{array} \right)_{(1,y,z)} \\
 \star \circ (f+g) \otimes h &= \left\{ \left((0, x), z \right) \mapsto (0, f_0(x), h_0(z)) \mapsto (0, (f_0(x), h_0(z))), \right. \\
 &\quad \left. ((1, y), z) \mapsto (1, g_0(y), h_0(z)) \mapsto (1, (g_0(y), h_0(z))) \right\}, \\
 &\quad \left(\begin{array}{c} f_x \\ \oplus \\ h_x \end{array} \right)_{(0,x,z)} \quad \text{::} \quad \left(\begin{array}{c} f_y \\ \oplus \\ h_y \end{array} \right)_{(1,y,z)}
 \end{aligned}$$

Figure 4.15: Proof of naturality of the distributivity of \otimes over $+$

We define left distributivity $\star' : A \otimes (- + -) \rightarrow (A \otimes -) + (A \otimes -)$ as follows:

$$\star'(B, C) = A \otimes (B + C) \xrightarrow{\sigma}_{\overline{M}} (B + C) \otimes A \xrightarrow{\star}_{\overline{M}} B \otimes A + C \otimes A \xrightarrow{\sigma + \sigma}_{\overline{M}} A \otimes B + A \otimes C.$$

Lemma 4.2.3. For any morphism $f : A \otimes B \rightarrow C$, the following diagram commutes:

$$\begin{array}{ccc}
 (A + A) \otimes B & \xrightarrow{(id, id) \otimes id} & A \otimes B \\
 \downarrow \star & & \downarrow f \\
 A \otimes B + A \otimes B & & C \\
 \downarrow f + f & & \downarrow (id, id) \\
 C + C & \xrightarrow{(id, id)} & C
 \end{array}$$

Proof. Proof is shown in Figure 4.16

$$\begin{aligned}
 f \circ ((id_A, id_A) \otimes id_B) &= \left\{ \left((b, x), y \right) \mapsto (x, y) \mapsto f_0(x, y) \right\}, \\
 &\quad \left(f_{x,y} \circ (id_{A_x} \otimes id_{B_y}) \right)_{(x+x, y)} \\
 (id_C, id_C) \circ (f + f) \circ \star &= \left\{ \left((0, x), y \right) \mapsto (0, (x, y)) \mapsto (0, f_0(x, y)) \mapsto f_0(x, y) \right\} \\
 &\quad \left\{ \left((1, x), y \right) \mapsto (1, (x, y)) \mapsto (1, f_0(x, y)) \mapsto f_0(x, y) \right\}, \\
 &\quad \left(id_{C_0} \circ f_{(x,y)} \circ (id_{A_x} \otimes id_{B_y}) \right)_{(x+x, y)} \\
 &\quad \left(id_{C_1} \circ f_{(x,y)} \circ (id_{A_x} \otimes id_{B_y}) \right)_{((1,x), y)}
 \end{aligned}$$

Figure 4.16: Proof of commutativity for the distributivity of \otimes over $+$

□

As an example, we can define true and false values as embeddings $f_{\text{tt}} : I \rightarrow I + I$ and $f_{\text{ff}} : I \rightarrow I + I$ as in Eq. (4.18), and use it as a switch as in Eq (4.19) for a control. Lemma 4.2.4 shows that the switch works as intended.

$$\begin{aligned} f_{\text{tt}} &::= (\{\emptyset \mapsto \text{tt}\}, (\text{id}_I)) \\ f_{\text{ff}} &::= (\{\emptyset \mapsto \text{ff}\}, (\text{id}_I)) \end{aligned} \quad (4.18)$$

where tt and ff are abbreviations for $(0, \emptyset)$ and $(1, \emptyset)$ respectively.

$$\text{switch}_{f,g}(b) ::= A \xrightarrow{l_A^{-1}} I \otimes A \xrightarrow{f_b \otimes \text{id}_A} (I+I) \otimes A \xrightarrow{\star} I \otimes A + I \otimes A \xrightarrow{l_{A+I}} A+A \xrightarrow{(f,g)} B \quad (4.19)$$

for any morphisms $f, g : A \rightarrow B$ and a boolean $b \in \{\text{tt}, \text{ff}\}$. Note that l_A refers to the natural transformation from the monoidal structure of $\overline{\overline{M}}$, which we let be equal to the identity morphism of A .

Lemma 4.2.4. *The following two diagrams commute: for any morphisms $f, g : A \rightarrow B$,*

$$\begin{array}{ccc} (A = I \otimes A) & \xrightarrow{f} & B \\ \downarrow f_{\text{tt}} \otimes \text{id}_A & & \uparrow (f,g) \\ (I+I) \otimes A & \xrightarrow{\star} & (I \otimes A + I \otimes A = A + A) \end{array} \quad \begin{array}{ccc} (A = I \otimes A) & \xrightarrow{g} & B \\ \downarrow f_{\text{ff}} \otimes \text{id}_A & & \uparrow (f,g) \\ (I+I) \otimes A & \xrightarrow{\star} & (I \otimes A + I \otimes A = A + A) \end{array}$$

Proof. Let's let $f = (f_0, (f_x))$ and $g = (g_0, (g_x))$

$$\begin{aligned} (f, g) \circ \star \circ (f_{\text{tt}} \otimes \text{id}_A) &= (\{\{\emptyset, x\} \mapsto (\text{tt}, x) \mapsto (0, (\emptyset, x)) \mapsto f_0(x)\}, (f_x)) = f \\ (f, g) \circ \star \circ (f_{\text{ff}} \otimes \text{id}_A) &= (\{\{\emptyset, x\} \mapsto (\text{ff}, x) \mapsto (1, (\emptyset, x)) \mapsto g_0(x)\}, (g_x)) = g \end{aligned}$$

□

4.3 . Benton's linear/non-linear category

Benton's linear/non-linear model [12] has been used in several models of quantum programming languages [76, 55, 24]. The model is based on the adjunction between a symmetric monoidal closed category and a cartesian closed category.

The adjunction gives us a comonad which is called !-comonad and lets us interpret the quantum circuit types and the circuit operators—box and un-box. In this section, we introduce the definition of the comonad based on the concrete category $\overline{\overline{M}}$.

4.3.1 . !-Comonad

Following the cases of [55, 24], we define an adjunction from the functors $p : \mathbf{Set} \rightarrow \overline{\overline{M}}$ and $b : \overline{\overline{M}} \rightarrow \mathbf{Set}$ between the symmetric monoidal closed category $\overline{\overline{M}}$ and the cartesian closed category \mathbf{Set} as shown in Figure 4.17.

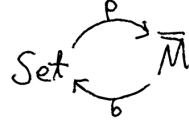


Figure 4.17: Adjunction for !-comonad

Definition of functors p and b

We begin with the functors between the category **Set** and $\overline{\overline{M}}$.

- functor $p : \mathbf{Set} \rightarrow \overline{\overline{M}}$
 - for object $X : p(X) = (X, (I)_X)$; and
 - for morphism $f : X \rightarrow Y : p(f) = (f, (\text{id}_I)_X) : (X, (I)_X) \rightarrow_{\overline{\overline{M}}} (Y, (I)_Y)$.

It can be seen that

$$p(g \circ f) = (g \circ f, (\text{id}_I)_X) = p(g) \circ p(f), \text{ and } p(\text{id}_X) = (\text{id}_X, (\text{id}_I)_X) = \text{id}_{p(X)}.$$

- functor $b : \overline{\overline{M}} \rightarrow \mathbf{Set}$
 - for object $A = (X, (A_x)) : b(A) = \sum_{x \in X} \overline{\overline{M}}(I, A_x)$ where $\overline{\overline{M}}(I, A_x)$ is the set of morphisms between the objects I and A_x of the category $\overline{\overline{M}}$ and $\sum_{x \in X} \overline{\overline{M}}(I, A_x)$ is the disjoint union of all such sets over X ; and
 - for morphism $f = (f_0, (f_x)) : (X, (A_x)) \rightarrow_{\overline{\overline{M}}} (Y, (B_y)) :$

$$b(f) = \{(x, p : I \rightarrow_{\overline{\overline{M}}} A_x) \mapsto (f_0(x), f_x \circ p)\} : \sum_{x \in X} \overline{\overline{M}}(I, A_x) \rightarrow \sum_{y \in Y} \overline{\overline{M}}(I, B_y).$$

Again, it can be seen that

- $b((g_0, (g_y)) \circ (f_0, (f_x))) = \{(x, f : I \rightarrow_{\overline{\overline{M}}} A_x) \mapsto ((g_0 \circ f_0)(x), f_{f_0(x)} \circ f_x \circ f)\} = b(g_0, (g_y)) \circ b(f_0, (f_x))$ and
- $b(\text{id}_X, (\text{id}_I)_X) = \{(x, f : I \rightarrow_{\overline{\overline{M}}} I) \mapsto (x, f)\} = \text{id}_{b(X, (A_x))}$.

Symmetric monoidal structure of $\overline{\overline{M}}$ and **Set**

As shown in Section 4.2, the category $\overline{\overline{M}}$ is a symmetric monoidal closed category. Moreover, we can show that the category **Set** is also a symmetric monoidal category with the following structure $(\mathbf{Set}, \times, \{\emptyset\}, \alpha, l, r, \sigma)$ where \times is the cartesian product, $\{\emptyset\}$ is a singleton set of \emptyset and the natural transformations are defined as below:

- $\alpha(X, Y, Z) = \{((x, y), z) \mapsto (x, (y, z))\} : (X \times Y) \times Z \rightarrow X \times (Y \times Z)$

- $l(X) = \{(\emptyset, x) \mapsto x\} : \{\emptyset\} \times X \rightarrow X$
- $r(X) = \{(x, \emptyset) \mapsto x\} : X \times \{\emptyset\} \rightarrow X$
- $\sigma(X, Y) = \{(x, y) \mapsto (y, x)\} : X \times Y \rightarrow Y \times X$

where the commute diagrams for naturality can be shown. Also, it can be shown that the structure satisfies the coherence conditions for the symmetric monoidal categories.

Monoidal functors (p, m) and (b, n)

In this setting, following the Benton's paper [12], we define the monoidal functors (p, m) and (b, n) as below. The monoidal functors mean that the functor preserves the symmetric monoidal structures of the two symmetric monoidal categories. We can show that the functors p and b are monoidal functors with the following definition of natural transformations m and n :

- $(p : \mathbf{Set} \rightarrow \overline{\overline{M}}, m) :$
 - $m_I = (\text{id}_{\{\emptyset\}}, (\text{id}_I)) : (\{\emptyset\}, (I)) \rightarrow_{\overline{\overline{M}}} (\{\emptyset\}, (I))$
 - $m_{X,Y} = \text{id}_{(X \times Y, (I)_{X \times Y})} : (X, (I)_X) \otimes (Y, (I)_Y) \rightarrow (X \times Y, (I)_{X \times Y})$,
where $(X, (I)_X) \otimes (Y, (I)_Y) = (X \times Y, (I)_{X \times Y})$

It can be shown that the following diagram of naturality commutes:

$$\begin{array}{ccc} (X, (I)_X) \otimes (Y, (I)_Y) & \xrightarrow{m_{X,Y}} & (X \times Y, (I)_{X \times Y}) \\ \downarrow p(f) \otimes p(g) & & \downarrow p(f \times g) \\ (X', (I)_{X'}) \otimes (Y', (I)_{Y'}) & \xrightarrow{m_{X',Y'}} & (X' \times Y', (I)_{X' \times Y'}) \end{array}$$

- $(b : \overline{\overline{M}} \rightarrow \mathbf{Set}, n) :$
 - $n_I = \{(\emptyset \mapsto \text{id}_I)\} : \{\emptyset\} \rightarrow \overline{\overline{M}}(I, I)$
 - $n_{A,B} = \{((x, f_x), (y, g_y)) \mapsto ((x, y), f_x \otimes g_y)\} : \sum_{x \in X} \overline{\overline{M}}(I, A_x) \times \sum_{y \in Y} \overline{\overline{M}}(I, B_y) \rightarrow \sum_{(x,y) \in X \times Y} \overline{\overline{M}}(I, A_x \otimes B_y)$

It can be shown that the following diagram of naturality commutes:

$$\begin{array}{ccc} \sum_{x \in X} \overline{\overline{M}}(I, A_x) \times \sum_{y \in Y} \overline{\overline{M}}(I, B_y) & \xrightarrow{n_{A,B}} & \sum_{(x,y) \in X \times Y} \overline{\overline{M}}(I, A_x \otimes B_y) \\ \downarrow b(f) \otimes b(g) & & \downarrow b(f \otimes g) \\ \sum_{x' \in X'} \overline{\overline{M}}(I, A'_{x'}) \times \sum_{y' \in Y'} \overline{\overline{M}}(I, B'_{y'}) & \xrightarrow{n_{A',B'}} & \sum_{(x',y') \in X' \times Y'} \overline{\overline{M}}(I, A'_{x'} \otimes B'_{y'}) \end{array}$$

There are coherence conditions monoidal functors [12]. They can be shown according to our definitions.

Natural transformations $\eta : \mathbf{1}_{\text{Set}} \rightarrow b \circ p$ and $\epsilon : p \circ b \rightarrow \mathbf{1}_{\overline{M}}$

Next, we define the natural transformations $\eta : \mathbf{1}_{\text{Set}} \rightarrow b \circ p$ and $\epsilon : p \circ b \rightarrow \mathbf{1}_{\overline{M}}$ for the adjunction (p, b, η, ϵ) .

- $\eta : \mathbf{1}_{\text{Set}} \rightarrow b \circ p$

We define it for a set X ,

$$\eta(X) = \{x \mapsto (x, \text{id}_I)\} : X \rightarrow (b \circ p)(X).$$

It satisfies the following commute diagram of naturality:

$$\begin{array}{ccc} X & \xrightarrow{\eta(X)} & (b \circ p)(X) \\ \downarrow f & & \downarrow (b \circ p)(f) \\ Y & \xrightarrow{\eta(Y)} & (b \circ p)(Y) \end{array}$$

- $\epsilon : p \circ b \rightarrow \mathbf{1}_{\overline{M}}$

We define, for an object $A = (X, (A_x)) \in \text{Obj}(\overline{M})$,

$$\epsilon(A) = (\{(x, p_x) \mapsto x \mid (x, p_x) \in \sum_{x \in X} \overline{M}(I, A_x)\}, (p_x)_{(x, p_x)}) : (p \circ b)(f) \rightarrow_{\overline{M}} A.$$

We can show that the following commute diagram holds:

$$\begin{array}{ccc} (p \circ b)(A) & \xrightarrow{\epsilon(A)} & A \\ \downarrow (p \circ b)(f) & & \downarrow f \\ (p \circ b)(A') & \xrightarrow{\epsilon(A')} & A' \end{array}$$

Moreover, we can show that η and ϵ are monoidal natural transformations by showing the following coherence conditions:

- $\eta : \mathbf{1}_{\text{Set}} \rightarrow (b \circ p)$

$$\begin{array}{ccc} X \times Y & \xrightarrow{\text{id}} & X \times Y \\ \downarrow \eta(X) \times \eta(Y) & & \downarrow \eta(X \times Y) \\ (b \circ p)(X) \times (b \circ p)(Y) & \xrightarrow{b(m_{X,Y}) \circ n(p(X), p(Y))} & (b \circ p)(X \times Y) \end{array} \quad \begin{array}{ccc} \{\emptyset\} & \xrightarrow{\eta(\{\emptyset\})} & (b \circ p)(\{\emptyset\}) \\ \text{id} \uparrow & \nearrow b(m_I) \circ n_I & \\ \{\emptyset\} & & \end{array}$$

- $\epsilon : (p \circ b) \rightarrow \mathbf{1}_{\overline{M}}$

$$\begin{array}{ccc} (p \circ b)(A) \otimes (p \circ b)(B) & \xrightarrow{p(n_{A,B}) \circ m(b(A), b(B))} & (p \circ b)(A \otimes B) \\ \downarrow \epsilon(A) \otimes \epsilon(B) & & \downarrow \epsilon(A \otimes B) \\ A \otimes B & \xrightarrow{\text{id}} & A \otimes B \end{array} \quad \begin{array}{ccc} (p \circ b)(I) & \xrightarrow{\epsilon(I)} & I \\ \uparrow p(n_I) \circ m_I & \swarrow \text{id} & \uparrow \\ I & & I \end{array}$$

Finally, we can show that (p, b, η, ϵ) is an adjunction by showing the following coherence conditions:

- $(b \circ \epsilon) \circ (\eta \circ b) = \text{id}_{b(-)}$
For any object A in \overline{M} , $b(\epsilon(A)) \circ \eta(b(A)) = \text{id}_{b(A)}$.
- $(\epsilon \circ p) \circ (p \circ \eta) = \text{id}_{p(-)}$
For any set X , $\epsilon(p(X)) \circ p(\eta(X)) = \text{id}_{p(X)}$.

4.3.2 . Symmetric monoidal comonad $(!, \epsilon : ! \rightarrow \mathbf{1}_{\overline{M}}, \delta : ! \rightarrow !!)$ with $\pi_{A,B}$ and π_I

From [12], a comonad $(! = p \circ b, \epsilon : ! \rightarrow \mathbf{1}_{\overline{M}}, \delta : ! \rightarrow !!)$ can be obtained from the natural transformation ϵ and $\delta = p \circ \eta \circ b$. The $!$ -comonad comes with the following properties of comonad:

$$\begin{array}{ccc} !A & \xleftarrow{\epsilon(!A)} & !!A & \xrightarrow{!(\epsilon(A))} & !A \\ & \searrow \text{id} & \uparrow \delta(A) & \nearrow \text{id} & \\ & & !A & & \end{array} \qquad \begin{array}{ccc} !!!A & \xleftarrow{!(\delta(A))} & !!A \\ \uparrow \delta(!A) & & \uparrow \delta(A) \\ !!A & \xleftarrow{\delta(A)} & !A \end{array}$$

Lemma 4.3.1. *There are natural transformations $(\epsilon \circ !)$ and δ between $!$ and $!!$, i.e. for any object $A \in \text{Obj}(\overline{M})$, there are morphisms $\epsilon(!A) : !!A \rightarrow !A$ and $\delta(A) : !A \rightarrow !!A$ which satisfy the following conditions:*

$$\begin{aligned} (\epsilon \circ !) \circ \delta &= \text{id}_! \\ \delta \circ (\epsilon \circ !) &\neq \text{id}_{!!}. \end{aligned}$$

Proof. Proof is shown in Figure 4.18.

$$\begin{aligned} (\epsilon \circ !) \circ \delta &= (\epsilon \circ p \circ b) \circ (p \circ \eta \circ b) = \left(\{ (x, f_x) \mapsto (x, f_x) \}, (id_x \circ id_x) \right) \\ &\quad : ! \rightarrow !! \rightarrow ! \\ \delta \circ (\epsilon \circ !) &= (p \circ \eta \circ b) \circ (\epsilon \circ p \circ b) = \left(\{ (x, f_x, m) \mapsto (x, f_x, id_x) \}, (id_x \circ m) \right) \\ &\quad \neq \text{id} \\ &\quad : !! \rightarrow ! \rightarrow !! \\ (p \circ \eta \circ b)(A) &= (p \circ \eta)(\sum_{x \in X} \overline{M}(I, A_x)) \\ &= P(\{ (x, f_x) \mapsto ((x, f_x), id_x) \}) \\ &= \left(\{ (x, f_x) \mapsto ((x, f_x), id_x) \}, (id_x)_{(x, f_x)} \right) \\ (p \circ b)A &= P(\sum_{x \in X} \overline{M}(I, A_x)) = \left(\sum_{x \in X} \overline{M}(I, A_x), (I) \right) \\ (\epsilon \circ p \circ b)(A) &= \epsilon(\sum_{x \in X} \overline{M}(I, A_x), (I)) = \left(\{ (x, f_x, m) \mapsto (x, f_x) \}, (m) \right) \end{aligned}$$

Figure 4.18: Proof of the lemma on the natural transformation between $!$ and $!!$

□

Next, the natural transformation $\pi_{A,B}$ and a morphism π_I are defined as follows:

- $\pi_I = (\{\emptyset \mapsto (\emptyset, \text{id}_I)\}, (\text{id}_I)) : I \rightarrow !I$
- $\pi(A, B) = (\{((x, m_x : I \rightarrow A_x), (y, m_y : I \rightarrow B_y)) \mapsto ((x, y), m_x \otimes m_y)\}, (\text{id}_I)) : (!A) \otimes (!B) \rightarrow !(A \otimes B)$ where the following diagram commutes :

$$\begin{array}{ccc} !A \otimes !B & \xrightarrow{\pi(A, B)} & !(A \otimes B) \\ \downarrow (!f) \otimes (!g) & & \downarrow !(f \otimes g) \\ !A' \otimes !B' & \xrightarrow{\pi(A', B')} & !(A' \otimes B') \end{array}$$

Additionally, we can show the following commute diagrams for monoidal comonad :

$$\begin{array}{ccc} !A \otimes !B & \xrightarrow{\pi(A, B)} & !(A \otimes B) \\ \searrow \epsilon \otimes \epsilon & & \swarrow \epsilon \\ & A \otimes B & \\ \downarrow \delta \otimes \delta & & \downarrow \delta \\ !!A \otimes !!B & \xrightarrow{\pi(!A, !B)} & !(A \otimes B) \xrightarrow{! \pi(A, B)} & !!(A \otimes B) \end{array} \quad \begin{array}{ccc} I & \xrightarrow{\pi_I} & !I \\ \searrow \text{id}_I & & \swarrow \epsilon \\ & I & \\ \downarrow \pi_I & & \downarrow \delta_I \\ !I & \xrightarrow{! \pi_I} & !!I \end{array}$$

4.4 . Lifting Monad

According to Rios&Selinger, the category $\overline{\overline{M}}$ together with the structure sketched in Section 4.2 and the Benton's linear/non-linear category described in Section 4.3 forms a model of Proto-Quipper-M. We shall now see how our concrete construction can also support dynamic lifting, therefore forming a model of Proto-Quipper-L.

The main problem consists in lifting a branching sitting inside a quantum channel —i.e., inside the category $\overline{\overline{M}}$ — to turn it into a coproduct on which one can act upon in the classical world, represented by the category \overline{M} . For instance, as in Remark 4.2, we need to lift a state-boolean into a parameter-boolean. Our strategy consists in defining a strong monad (F, μ, η, t) to capture the action of retrieving such a branching: a term featuring measurement (and dynamic lifting) is therefore represented within the Kleisli category $\overline{\overline{M}}_F$, following Moggi's [44] view on side-effects.

The functor $F : \overline{\overline{M}} \rightarrow \overline{\overline{M}}$ is defined as follows. For an object $A = (X, (A_x))$, we define $F(A) = (\text{mset}(X), ([\boxplus_{x \in l} A_x^\otimes])_{l \in \text{mset}(X)})$, where $\text{mset}(X)$ is the set of multisets of X , while for a morphism $f = (f_0, (f_x)) : A \rightarrow B$ we set $F(f) = (g_0 : \text{mset}(X) \rightarrow \text{mset}(Y), g_l : [\boxplus_{x \in l} A_x^\otimes] \rightarrow [\boxplus_{y \in g_0(l)} B_y^\otimes])$, where $g_0 = \{[x_0, \dots, x_n] \mapsto [f_0(x_0), \dots, f_0(x_n)]\}$ and where g_l is defined as shown on the right.

Example 4.4.1. The lifting of the state boolean b_s of Remark 4.2 to the parameter boolean b_p is then a $\overline{\overline{M}}$ -map $lb : b_s \rightarrow F(b_p)$, where $F(b_p)$ is $(mset\{tt, ff\}, (\boxplus_{x \in I} I)_l)$. The map lb is defined as $(lb_0, (lb_x)_{x \in \{\emptyset\}})$ where $lb_0 : \{\emptyset\} \rightarrow mset\{tt, ff\}$ sends \emptyset to $[tt, ff]$, and where $lb_\emptyset : I \boxplus I \rightarrow I \boxplus I$ is simply defined as the identity. In the other direction, the $\overline{\overline{M}}$ -map $b_p \rightarrow b_s$ consists of the constant set-function on \emptyset together with the injections $I \rightarrow I \boxplus I$ discussed in Section 4.1.

4.4.1 . Definition of Branching computation monad (F, μ, η, t)

Our categorical model is equipped a strong monad (F, μ, η, t) , which is a monad (F, μ, η) with a tensorial strength t . The strong monad models the computation that creates a non-deterministic branch. The monad provides us a Kleisli category $\overline{\overline{M}}_F$ where the computation is interpreted as a morphism [44].

We begin with the definition of a functor $F : \overline{\overline{M}} \rightarrow \overline{\overline{M}}$ in Definition 4.4.1.

Definition 4.4.1 (Functor F for the branching computation monad). Functor is defined over the category $\overline{\overline{M}}$ as follows.

- for object $A = (X, (A_x))$: $F(A) = (mset(X), ([\boxplus_{x \in I} A_x^\otimes])_{l \in mset(X)})$, where **mset**(X) is the set of multisets of X ;
- for morphism $f = (f_0, (f_x)) : A \rightarrow B$:

$$F(f) = (g_0 : mset(X) \rightarrow mset(Y), g_l : [\boxplus_{x \in l} A_x^\otimes] \rightarrow [\boxplus_{y \in g_0(l)} B_y^\otimes]),$$

where

$$g_0 = \{[x_0, \dots, x_n] \mapsto [f_0(x_0), \dots, f_0(x_n)]\} \text{ and } g_l :=$$



The functor F preserves the identity morphism and the composition of morphisms:

$$F(\text{id}_A) = \text{id}_{F(A)}, \text{ and } F(g \circ f) = F(g) \circ F(f).$$

Next, the unit η and the multiplication μ of the monad are defined as in Definition 4.4.2.

Definition 4.4.2 (Unit and multiplication of branching monad). Unit η and multiplication μ are defined as natural transformations as follows.

- $\eta : \mathbf{1}_{\overline{M}} \rightarrow F$

For an object $A \in \text{Obj}(\overline{M})$, $\eta(A) = (\{x \mapsto [x]\}, (\otimes : A_x \rightarrow A_x^\otimes)_X) : A \rightarrow_{\overline{M}} F(A)$. The following commute diagram for the naturality can be shown:

$$\begin{array}{ccc} A & \xrightarrow{\eta(A)} & F(A) \\ \downarrow f & & \downarrow F(f) \\ A' & \xrightarrow{\eta(A')} & F(A') \end{array}$$

- $\mu : F^2 \rightarrow F$

For an object $A \in \text{Obj}(\overline{M})$,

$$\begin{aligned} \mu(A) = & (\{[l_1, \dots, l_n] \mapsto l_1 + \dots + l_n\}, (\text{id}_{\boxplus_{l \in L} \boxplus_{x \in l} A_x^\otimes}) \\ & : \boxplus_{l \in L} [\boxplus_{x \in l} A_x^\otimes] \rightarrow \boxplus_{x \in l_1 + \dots + l_n} A_x^\otimes = F^2(A) \rightarrow_{\overline{M}} F(A), \end{aligned}$$

where

The commute diagram of the naturality can be shown:

$$\begin{array}{ccc} F^2(A) & \xrightarrow{\mu(A)} & F(A) \\ \downarrow F^2(f) & & \downarrow F(f) \\ F^2(A') & \xrightarrow{\mu(A')} & F(A') \end{array}$$

Lemma 4.4.1. Given the definition above, (F, μ, η) is a monad, i.e. the following diagrams commute:

$$\begin{array}{ccc} F^3(A) & \xrightarrow{\mu(F(A))} & F^2(A) \\ \downarrow F(\mu(A)) & & \downarrow \mu(A) \\ F^2(A) & \xrightarrow{\mu(A)} & F(A) \end{array} \quad \text{and} \quad \begin{array}{ccc} F(A) & \xrightarrow{\eta(F(A))} & F^2(A) & \xleftarrow{F(\eta(A))} & F(A) \\ & \searrow \text{id} & \downarrow \mu(A) & \swarrow \text{id} & \\ & & F(A) & & \end{array}$$

Proof. The commute diagrams can be shown as in Figure 4.19. □

In addition, Lemma 4.4.2 states a commute relation over the coproduct in \overline{M} and the natural transformation μ .

$$\begin{aligned}
\mathcal{M}(A) \circ F(\eta(A)) &= (\{[x_1, \dots, x_n] \mapsto [x_1, \dots, x_n]\} \mapsto [x_1, \dots, x_n]), \\
\mathcal{M}(A) \circ \mathcal{M}(FA) &= (\{[l_{11}, \dots, l_{1n}] \mapsto [l_{11}, \dots, l_{1n}]\} \mapsto [l_{11}, \dots, l_{1n}], (id)) \\
\mathcal{M}(A) \circ F(\mathcal{M}(A)) &= (\{[l_{11}, \dots, l_{1n}] \mapsto [l_{11}, \dots, l_{1n}]\} \mapsto [l_{11}, \dots, l_{1n}], (id))
\end{aligned}$$

Figure 4.19: Proof of the coherence conditions for branching monad

Lemma 4.4.2. *The following diagram commutes:*

$$\begin{array}{ccccc}
F(F^2Z + F^2Z) & \xrightarrow{F(id, id)} & F^3Z & \xrightarrow{\mu} & F^2Z \\
\downarrow F(\mu + \mu) & & & & \downarrow \mu \\
F(FZ + FZ) & \xrightarrow{F(id, id)} & F^2Z & \xrightarrow{\mu} & FZ
\end{array}$$

Proof. The proof is shown in Figure 4.20.

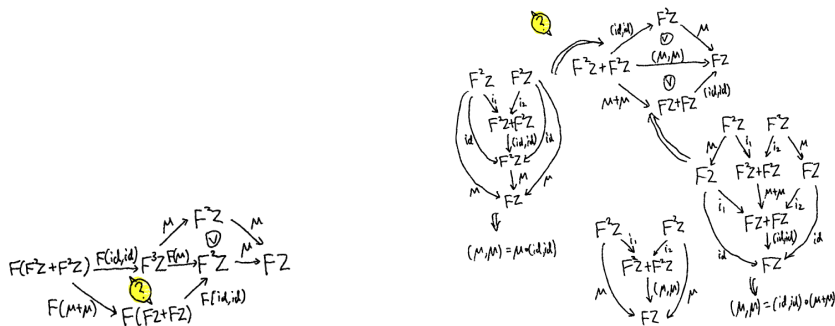


Figure 4.20: Proof of commutativity between coproduct and branching monad

□

Next, we define the tensorial strength as in Definition 4.4.3 which makes the branching monad a strong monad [44, 76].

Definition 4.4.3 (Tensorial strength t of branching monad). Tensorial strength is the natural transformation t defined as follows.

- For any objects $A, B \in \text{Obj}(\overline{M})$,

$$t_{A,B} = (\{([x_1, \dots, x_n], y) \mapsto [(x_1, y), \dots, (x_n, y)]\}, (\begin{array}{c} \uparrow \text{Hom}(A_x \otimes B_y) \\ \boxed{\begin{array}{c} \oplus \\ x_1 \end{array}} \begin{array}{c} \uparrow A_x \otimes B_y \\ \otimes \\ \downarrow \\ \oplus \\ y \end{array} \\ \uparrow A_x \quad \uparrow B_y \end{array}))$$

$$: F(A) \otimes B \rightarrow_{\overline{M}} F(A \otimes B).$$

The natural transformation satisfies the following commute diagram:

$$\begin{array}{ccc} F(A) \otimes B & \xrightarrow{t_{A,B}} & F(A \otimes B) \\ \downarrow F(f) \otimes g & & \downarrow F(f \otimes g) \\ F(A') \otimes B' & \xrightarrow{t_{A',B'}} & F(A' \otimes B') \end{array}$$

The naturality of tensorial strength t can be shown as follows. For $f = (f_0, (f_x)_x) : A \rightarrow A'$ and $g = (g_0, (g_y)_y) : B \rightarrow B'$,

$$F(f \otimes g) \circ t_{A,B} =$$

$$(\{([x_1, \dots, x_n], y) \mapsto [(f_0(x_1), g_0(y)), \dots, (f_0(x_n), g_0(y))]\}, (\begin{array}{c} \boxed{\begin{array}{c} \oplus \\ x_1 \end{array}} \begin{array}{c} \uparrow A_x \otimes B_y \\ \otimes \\ \downarrow \\ \oplus \\ y \end{array} \\ \uparrow A_x \quad \uparrow B_y \end{array}))$$

$$t_{A',B'} \circ (F(f) \otimes g) =$$

$$(\{([x_1, \dots, x_n], y) \mapsto [(f_0(x_1), g_0(y)), \dots, (f_0(x_n), g_0(y))]\}, (\begin{array}{c} \boxed{\begin{array}{c} \oplus \\ x_1 \end{array}} \begin{array}{c} \uparrow A_x \otimes B_y \\ \otimes \\ \downarrow \\ \oplus \\ y \end{array} \\ \uparrow A_x \quad \uparrow B_y \end{array}))$$

where $F(f \otimes g) \circ t_{A,B} = t_{A',B'} \circ (F(f) \otimes g)$ follows from Figure 4.21.

Moreover, Lemma 4.4.3 shows that Definition 4.4.3 satisfies the conditions for tensorial strengths.

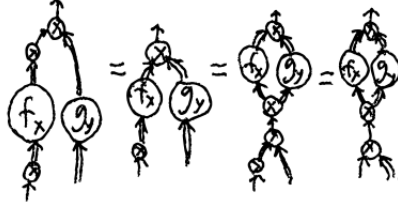


Figure 4.21: Proof of the naturality of the tensorial strength of branching monad

Lemma 4.4.3. *Tensorial strength t satisfies the following commute diagrams.*

$$\begin{array}{ccc}
 F(A) \otimes I & \xrightarrow{t_{A, id_I}} & F(A \otimes I) \\
 & \searrow^{r_{F(A)}} & \downarrow F(r_A) \\
 & & F(A)
 \end{array}
 ,
 \begin{array}{ccc}
 A \otimes B & \xrightarrow{id_{A \otimes B}} & A \otimes B \\
 \downarrow \eta(A) \otimes id_B & & \downarrow \eta(A \otimes B) \\
 F(A) \otimes B & \xrightarrow{t_{A, B}} & F(A \otimes B) \\
 \mu(A) \otimes id_B \uparrow & & \mu(A \otimes B) \uparrow \\
 F^2(A) \otimes B & \xrightarrow{t_{A', B'}} & F(F(A) \otimes B) \xrightarrow{F(t_{A, B})} F^2(A \otimes B)
 \end{array}
 ,$$

$$\begin{array}{ccc}
 F(A) \otimes (B \otimes C) & \xrightarrow{t_{A, B \otimes C}} & F(A \otimes (B \otimes C)) \\
 \downarrow \alpha_{F(A), B, C}^{-1} & & \downarrow F(\alpha_{A, B, C}^{-1}) \\
 (F(A) \otimes B) \otimes C & \xrightarrow{t_{A, B} \otimes id_C} & F(A \otimes B) \otimes C \xrightarrow{t_{A \otimes B, C}} F((A \otimes B) \otimes C)
 \end{array}$$

Proof. The proof of each commute diagram can be found in Figure 4.22. \square

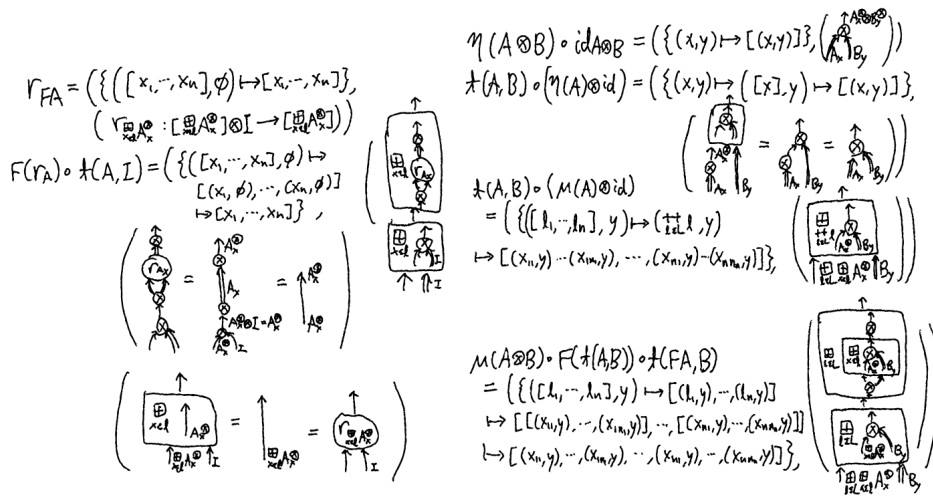
For convenience, we define another tensorial strength $\psi : F(-) \otimes F(-) \rightarrow F(- \otimes -)$ from the tensorial strength t as in Eq.(4.20).

$$\psi_{A, B} = \begin{array}{ccc}
 F(A) \otimes F(B) & \xrightarrow{t_{A, F(B)}} & F(A \otimes F(B)) \xrightarrow{F(\sigma)} F(F(B) \otimes A) \\
 & & \downarrow F(t_{B, A}) \\
 F(A \otimes B) & \xleftarrow{\mu} & F(F(A \otimes B)) \xleftarrow{F^2(\sigma)} F(F(B \otimes A))
 \end{array} \quad (4.20)$$

Lemma 4.4.4. *The $\psi_{A, B}$ is equivalently represented as the following morphism in $\overline{\overline{M}}$.*

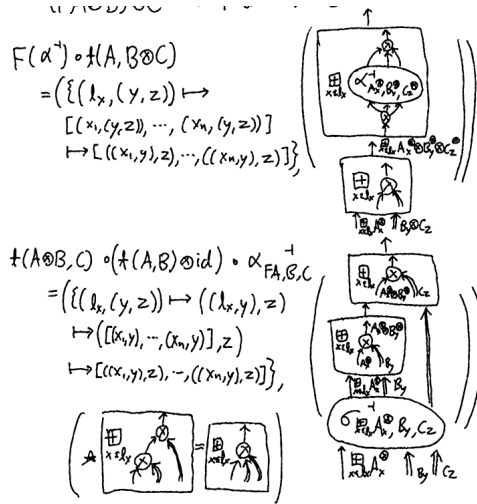
$$\psi_{A, B} = \begin{array}{ccc}
 F(A) \otimes F(B) & \xrightarrow{t_{A, F(B)}} & F(A \otimes F(B)) \xrightarrow{F(\sigma)} F(F(B) \otimes A) \\
 \xrightarrow{F(t)} & F(F(B \otimes A)) & \xrightarrow{\mu} F(B \otimes A) \xrightarrow{F(\sigma)} F(A \otimes B)
 \end{array}$$

Then, we show the following properties of tensorial strength in Lemma 4.4.5.



(a) Commute diagram 1

(b) Commute diagram 2



(c) Commute diagram 3

Figure 4.22: Proof of the commute diagrams for the tensorial strength of monad

Lemma 4.4.5. *The following diagrams commute:*

$$\begin{array}{ccc}
 A \otimes B & & F^2(A) \otimes F(B) \xrightarrow{\mu \otimes id_{F(B)}} F(A) \otimes F(B) \\
 \downarrow \eta \otimes \eta & \searrow \eta & \downarrow \psi \\
 F(A) \otimes F(B) & \xrightarrow{\psi} & F(A \otimes B)
 \end{array}
 \quad \text{and} \quad
 \begin{array}{ccc}
 F^2(A) \otimes F(B) & & F(A) \otimes F(B) \\
 \downarrow \psi & & \downarrow \psi \\
 F(F(A) \otimes B) & \xrightarrow{F(\dagger)} & F^2(A \otimes B) \xrightarrow{\mu} F(A \otimes B)
 \end{array}$$

Proof. Proofs of the commute diagrams are shown in Figure 4.23

□

4.4.2 . Kleisli category \overline{M}_F of the monad F

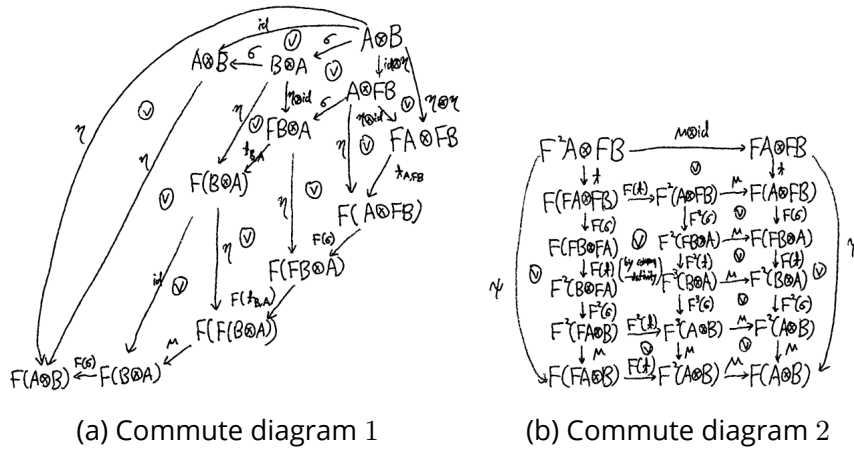


Figure 4.23: Proof of the properties of tensorial strength ψ

The strong monad (F, μ, η, t) gives us the Kleisli category $\overline{\overline{M}}_F$ whose object is the object of the category \overline{M} and the homset $\text{Hom}(A, B)$ of which is the homset $\text{Hom}(A, F(B))$ in \overline{M} . As the type system is interpreted in the category of the Kleisli category of some computation monad in [44], we interpret our type system in the category $\overline{\overline{M}}_F$.

4.4.3 . Natural isomorphism between $!F$ and $F!$

Lastly, we define the following natural transformation ϕ between the functors $! \circ F$ and $F \circ !$.

Definition 4.4.4. For object A in $\overline{\overline{M}}$, $\phi(A) = (\{(l, f_l : I \rightarrow [\boxplus_{x \in I} A_x^\otimes]) \mapsto [(x_1, f_{x_1}), \dots, (x_n, f_{x_n})], (\text{id}_I)\} : !F(A) \rightarrow_{\overline{\overline{M}}} F!(A)$ where



Note that the naturality of ϕ can be shown .

The inverse of the morphism $\phi(A)$ can be explicitly shown by the following.

$$\phi^{-1}(A) = (\{[(x_1, f_{x_1}), \dots, (x_n, f_{x_n})] \mapsto (l, f_l)\}, (\text{id}_I)).$$

Lemma 4.4.6. The following diagrams commute:

$$\begin{array}{ccc} !A & \xrightarrow{\text{id}} & !A \\ \downarrow !(\eta(A)) & & \downarrow \eta(!A) \\ !F(A) & \xrightarrow{\phi(A)} & F!(A) \end{array} \quad \begin{array}{ccc} !F(A) & \xrightarrow{\phi(A)} & F!(A) \\ \searrow \epsilon(F(A)) & & \downarrow F(\epsilon(A)) \\ & & F(A) \end{array}$$

(a) Proof of the left property

$$\eta(!A) = \left(\left\{ (x, m_x) \mapsto [(x, m_x)] \right\}, (\otimes: I \rightarrow I^{\otimes}) \right)$$

$$\phi(A) \circ !(\eta(A)) = \left(\left\{ (x, m_x) \mapsto [(x], \otimes \circ m_x) \right\}, (id_I) \right)$$

$$\left(\begin{array}{l} \eta(A) = \left(\{x \mapsto [x]\}, (\otimes: A_x \rightarrow A_x^{\otimes})_x \right) \\ !(\eta(A)) = \left(\{ (x, m_x: I \rightarrow A_x) \mapsto [(x], \otimes \circ m_x) \}, (id_I) \right) \end{array} \right)$$

(b) Proof of the right property

$$F(\varepsilon(A)) \circ \phi(A) = \left(\left\{ (\lambda, f_\lambda: I \rightarrow [\prod_{x \in I} A_x^{\otimes}]) \right\}, \left(\begin{array}{c} \uparrow \prod_{x \in I} A_x^{\otimes} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \end{array} \right) \right)$$

$$\begin{aligned} &\mapsto \left\{ [(x_i, f_{x_i}), \dots, (x_{m_i}, f_{x_{m_i}})] \right\} \\ &\mapsto \left\{ [x_i, \dots, x_{m_i}] \right\}, \end{aligned}$$

$$F(\varepsilon(A)) = F(\left\{ (x, p_x) \mapsto x \right\}, (p_x)_{(x, p_x)}) = \left\{ [(x, f_x), \dots, (x_n, f_{x_n})] \mapsto [x_i, \dots, x_n] \right\}$$

$$\left(\begin{array}{c} \uparrow \prod_{x \in I} A_x^{\otimes} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \end{array} \right)$$

$$\varepsilon(F(A)) = \left(\left\{ (\lambda, f_\lambda: I \rightarrow [\prod_{x \in I} A_x^{\otimes}]) \mapsto \lambda \right\}, (f_\lambda)_{(x, f_\lambda)} \right)$$

$$F(A) = (\text{msref}(x), [\prod_{x \in I} A_x^{\otimes}]_{I \rightarrow \text{msref}(x)})$$

Figure 4.24: Proof of the properties of natural transformation ϕ between the branching monad and the !-comonad

Proof. Proofs of the commute diagrams are shown in Figure 4.24 where the following notation is used in the proof of the right diagram.

$$(\lambda, f_\lambda) \in \sum_{\lambda \in \text{msref}(x)} \overline{M}(I, [\prod_{x \in I} A_x^{\otimes}])$$

where $\lambda = [x_1, \dots, x_n]$

and $f_\lambda = \left(\begin{array}{c} \uparrow \prod_{x \in I} A_x^{\otimes} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \\ \downarrow \prod_{x \in I} A_x^{\otimes} \\ \text{[} \prod_{x \in I} A_x^{\otimes} \text{]} \end{array} \right)$

□

4.5 . Interpretation of the language

In this section, we introduce an interpretation of the language Proto-Quipper-L and its type system introduced in Chapter 3 within the Kleisli category $\overline{\overline{M}}_F$. It consists of the interpretation of each type, the interpretation of typing context, and the interpretation of typing derivation, each of which assigns objects or morphisms in the categorical model to each part of the type system. As customary, types are mapped to objects while typing derivations are mapped to morphisms. For the notation, we let $\llbracket - \rrbracket$ represent the categorical counterparts for each component of the type system. The interpretation $\llbracket A \rrbracket$ of a type A is directly built against the categorical structure. In specific, types are interpreted as object in the Kleisli category $\overline{\overline{M}}_F$ as follows:

$$\begin{aligned}
\llbracket I \rrbracket &= (\{\emptyset\}, (I)) & \llbracket \text{bool} \rrbracket &= (\{\text{tt}, \text{ff}\}, (I, I)) \\
\llbracket \text{qubit} \rrbracket &= (\{\emptyset\}, ([q])) & \llbracket A_a \multimap A_b \rrbracket &= \llbracket A_a \rrbracket \multimap_{\overline{M}_F} \llbracket A_b \rrbracket \\
\llbracket A_a \otimes B_b \rrbracket &= \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket & \llbracket !A \rrbracket &= !\llbracket A \rrbracket = (p \circ b) \llbracket A \rrbracket \\
\llbracket \text{QChan}(P, B) \rrbracket &= p(\overline{M}_F(\llbracket P \rrbracket, \llbracket A \rrbracket))
\end{aligned}$$

where $\llbracket A_a \rrbracket \multimap_{\overline{M}_F} \llbracket A_b \rrbracket$ means the internal hom $\llbracket A_a \rrbracket \multimap_{\overline{M}} F \llbracket A_b \rrbracket$ in the category \overline{M} and p and b refer to the functors between the category \overline{M} and the category Set from Benton's linear/non-linear category in Section 4.3.

For quantum channels, we follow Rios&Selinger's strategy by defining $\llbracket \text{QChan}(P, B) \rrbracket = p(\overline{M}_F(\llbracket P \rrbracket, \llbracket A \rrbracket))$. In our situation, the set $\overline{M}_F(A, B)$ is isomorphic to $\overline{M}(A, B)$ when A and B are state objects: in this situation, QChan -types indeed correspond to morphisms of the category \overline{M} , i.e., quantum channels. This fact allows us to interpret the box and unbox constants. The constant quantum channel constant is then just an encapsulation over Definition 4.1.2.

Finally, a typed configuration $!\Delta \vdash (Q, m) : A$ is interpreted as the composition of Q (i.e., we first "compute" Q) followed by the interpretation of M .

For the typing context, we do not consider the name of variables but the types. However, to represent the types in the typing context, we need some order of the type assignments. Here, we assume that there is a linear order of the variable names and use this order to interpret the typing context.

In particular, typing context is constructed in two ways. First, conventional typing context is a list of the typing assignments $(x : A)$ for variable x and a type A . We interpret it as the tensor product of the objects assigned to each type in the order of the ordering we gave to the variables. In specific, for a typing context $\Gamma = x_1 : A_1, \dots, x_k : A_k$, assuming the variables are ordered according to the index,

$$\llbracket \Gamma \rrbracket = \llbracket A_1 \rrbracket \otimes \dots \otimes \llbracket A_k \rrbracket.$$

Next, the branching typing context is interpreted as the coproduct of the objects assigned to the smaller branching typing contexts, namely:

$$\llbracket \gamma_1, \gamma_2 \rrbracket = \llbracket \gamma_1 \rrbracket + \llbracket \gamma_2 \rrbracket.$$

Given the interpretation of type and typing context, we interpret the typing derivation as a morphism in \overline{M}_F . Specific morphism corresponding to each typing derivation is constructed inductively over the typing rules by using the structures in the category.

In order to formally define the construction for each typing rule, let us first introduce how the parts of the language like circuit operators and function applications are going to be interpreted. Then, the interpretation for each typing rule can be defined concretely.

Example 4.5.1. *The example of non-trivial branching term in Example 1.1 has for interpretation a morphism $(\{\emptyset\}, (q)) \rightarrow (mset\{(\emptyset, \emptyset)\}, (q)_I)$ defined as $(f_0, (f_\emptyset))$ where $f_0(\emptyset) = [(\emptyset, \emptyset), (\emptyset, \emptyset)]$ and f_\emptyset is defined as the diagram shown in Figure 4.25. The bottom box-node represents the measurement $(I \boxplus I)$ being the result) and the upper one the test. The top result is a \boxplus -superposition of 2 copies of $q \otimes I$, as expected: these stand for the two “classical” possibilities.*

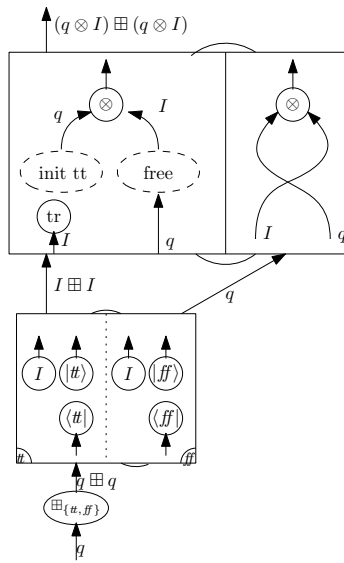


Figure 4.25: Diagram for the categorical interpretation of the non-trivial branching term in Example 1.1

4.5.1 . Interpretation of quantum channel types and the circuit operators, Box and Unbox

The adjunction between the category $\overline{\overline{M}}$ and the category Set in Figure 4.17 provides an interpretation of the quantum channel types and the box and unbox operators. As in [55], we interpret the quantum channel types $\text{QChan}(A, B)$ as an object $p(\overline{\overline{M}}_F(A, B)) = (\overline{\overline{M}}_F(A, B), (I))$ in $\overline{\overline{M}}_F$, which is the Kleisli category of the lifting monad F over $\overline{\overline{M}}$. Note that the object is a parameter object as in [55], which means that the object has the form of $(X, (I)_X)$ for some set X . Therefore, we can consider the quantum channel type $\text{QChan}(A, B)$ as a parameter of a set of diagrams from A to B . It makes sense when A and B are state objects which consist of objects in the category of diagrams $\overline{\overline{M}}$. However, since the output type B can be any type (while the

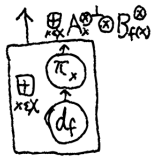
input type A is still limited to the state object), we generalize the isomorphism used in [55] between the object of quantum channel type $p(\overline{\overline{M}}_F(A, B))$ and the internal hom $\overline{\overline{M}}_F(A, B)$. Let us explain how we generalize the isomorphism.

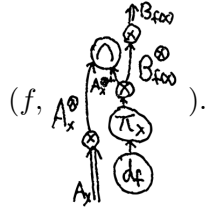
When we define the quantum channel types $\text{QChan}(A, B)$ as a parameter object, box and unbox can be interpreted based on an isomorphism between the set $b(A \multimap_{\overline{M}} B)$ and the homset $\overline{\overline{M}}(A, B)$. In specific, we can define an isomorphism as follows. First, note that

$$\begin{aligned} b(A \multimap_{\overline{M}} B) &= b(X \rightarrow Y, (\boxplus_{x \in X} (A_x \multimap B_{f(x)}))_{f \in X \rightarrow Y}) \\ &= \sum_{f \in X \rightarrow Y} \overline{\overline{M}}(I, \boxplus_{x \in X} (A_x \multimap B_{f(x)})) \\ &= \{(f, d_f) \mid f : X \rightarrow Y \text{ and } d_f : I \rightarrow [\boxplus_{x \in X} (A_x \multimap B_{f(x)})^{\otimes}]\} \end{aligned}$$

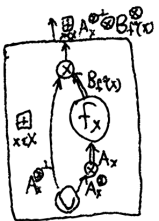
and that $\overline{\overline{M}}(A, B) = \{(f_0 : X \rightarrow Y, (f_x : A_x \rightarrow B_{f_0(x)})_X)\}$. Therefore, it reduces to find an isomorphism between the diagram d_f and the family of diagram $(f_x)_{x \in X}$. However, from the lemma 4.1.4, we know that the morphism d_f from I to $[\boxplus_{x \in X} (A_x \multimap B_{f(x)})^{\otimes}]$ is equal to a box node of the family of diagram $(\pi_x \circ d_f : I \rightarrow A_x)_{x \in X}$ which allow us to define the isomorphism as follows:

- $\text{iso}_{\rightarrow} : b(A \multimap_{\overline{M}} B) \rightarrow \overline{\overline{M}}(A, B)$:

Given $(f, d_f) \in b(A \multimap_{\overline{M}} B)$, where $d_f =$ , $\text{iso}_{\rightarrow}(f, d_f) =$



- $\text{iso}_{\leftarrow} : \overline{\overline{M}}(A, B) \rightarrow b(A \multimap_{\overline{M}} B)$:

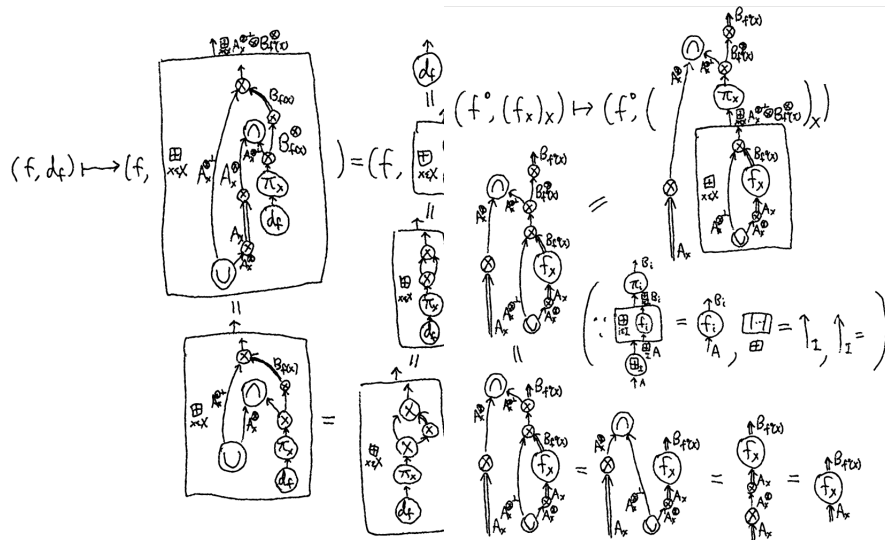
Given $(f_0, (f_x)_X) \in \overline{\overline{M}}(A, B)$, $\text{iso}_{\leftarrow}(f_0, (f_x)_X) = (f_0,$ )

Lemma 4.5.1. *The maps iso_{\rightarrow} and iso_{\leftarrow} are isomorphism. In specific, the fol-*

lowing diagrams commute:

$$\begin{array}{ccc}
 b(A \dashv_{\overline{M}} B) & \xrightarrow{\text{iso} \rightarrow} & \overline{\overline{M}}(A, B) \\
 \searrow \text{id} & & \downarrow \text{iso} \leftarrow \\
 & & b(A \dashv_{\overline{M}} B)
 \end{array}
 \quad \text{and} \quad
 \begin{array}{ccc}
 \overline{\overline{M}}(A, B) & & \\
 \downarrow \text{iso} \leftarrow & \searrow \text{id} & \\
 b(A \dashv_{\overline{M}} B) & \xrightarrow{\text{iso} \rightarrow} & \overline{\overline{M}}(A, B)
 \end{array}$$

Proof. The proof of the one-to-one correspondence of given isomorphisms is in Figure 4.26.



(a) Proof of the first commute diagram (b) Proof of the second commute diagram

Figure 4.26: Proof of the isomorphism between $b(A \dashv_{\overline{M}} B)$ and $\overline{\overline{M}}(A, B)$

□

Given the isomorphism, we can define the morphisms for box and unbox as morphisms in $\overline{\overline{M}}$ as follows:

$$\begin{aligned}
 \text{unbox} &= p(\overline{\overline{M}}(A, F(B))) \xrightarrow{p(\text{iso} \rightarrow)} (p \circ b)(A \dashv_{\overline{M}} F(B)) \xrightarrow{\epsilon(A \dashv_{\overline{M}} F(B))} (A \dashv_{\overline{M}} F(B)) \\
 \text{box} &= (p \circ b)(A \dashv_{\overline{M}} F(B)) \xrightarrow{p(\text{iso} \leftarrow)} p(\overline{\overline{M}}(A, F(B))) \xrightarrow{p(\eta(\overline{\overline{M}}(A, F(B))))} (p \circ b \circ p)(\overline{\overline{M}}(A, F(B))).
 \end{aligned}$$

Lemma 4.5.2. *The following diagram commutes:*

$$\begin{array}{ccc}
 p(\overline{\overline{M}}(A, F(B))) & \xleftarrow{\epsilon} & (p \circ b \circ p)(\overline{\overline{M}}(A, F(B))) \\
 \downarrow \text{unbox} & & \text{box} \uparrow \\
 A \multimap_{\overline{\overline{M}}} F(B) & \xleftarrow{\epsilon} & !(A \multimap_{\overline{\overline{M}}} F(B))
 \end{array}$$

Proof. Proof of the lemma can be found in Figure 4.27.

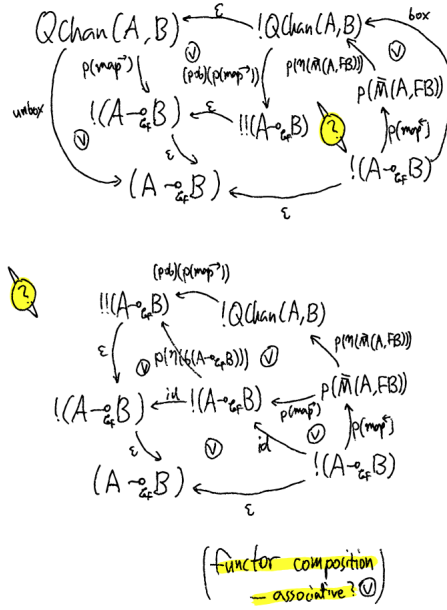


Figure 4.27: Proof of the commute diagram on box and unbox

□

As consequence, it follows that $\text{unbox} \circ \epsilon \circ \text{box} = \epsilon$.

The isomorphism between the sets $b(A \multimap_{\overline{\overline{M}}} B)$ and $\overline{\overline{M}}(A, B)$ is a slight generalization of the isomorphism in [55] in that A and B are need to be state objects, i.e. the object of the form $(\{\emptyset\}, (A_\emptyset))$, in their case. Note that when $A = (\{\emptyset\}, (A_\emptyset))$ and $B = (\{\emptyset\}, (B_\emptyset))$ are state objects, the set $b(A \multimap_{\overline{\overline{M}}} B)$ is equal to $\overline{\overline{M}}(I, A_\emptyset \multimap B_\emptyset)$ as follows:

$$\begin{aligned}
 A \multimap_{\overline{\overline{M}}} B &= (X \rightarrow Y, (\boxplus_{x \in X} (A_x \multimap B_{f(x)}))_{f: X \rightarrow Y}) \\
 b(A \multimap_{\overline{\overline{M}}} B) &= \sum_{f \in X \rightarrow Y} \overline{\overline{M}}(I, \boxplus_{x \in X} (A_x \multimap B_{f(x)})) \\
 &= \overline{\overline{M}}(I, A_\emptyset \multimap B_\emptyset).
 \end{aligned}$$

since $X = \{\emptyset\}$ and there is unique map from $\{\emptyset\}$ to $\{\emptyset\}$ which is the identity function. Then, we can obtain the following isomorphisms without the

necessity of the lemma 4.1.4 on the normal form of diagrams:

$$b(A \xrightarrow[\overline{M}]{} B) \cong \overline{M}(I, A_\emptyset \multimap B_\emptyset) \cong \overline{M}(A_\emptyset, B_\emptyset)$$

4.5.2 . Interpretation of the quantum channel constants

A quantum channel constant (p, Q, m) consists of a pattern p , a quantum channel Q , and a branching term m . A well typed quantum channel constant of type $!Q\text{Chan}(P, A)$ needs to satisfy that $p \models P$ and $v\text{Bind}(!\Delta, \text{out}(Q), m, A)$, where $!\Delta$ is a typing context. In this subsection, we introduce how to interpret a quantum channel and the relations $v\text{Bind}$.

First of all, we define a natural transformations called bif and merge for the measurement as in Table 4.5.


$\text{bif}(A) : A \xrightarrow[\overline{M}_F]{} A + A$	$\text{merge}(A, B) : F(A) + F(B) \xrightarrow[\overline{M}_F]{} A + B$
For an object $A = (X, (A_x))$, we let $\text{bif}(X, (A_x)) =$ $(\{x \mapsto [(0, x), (1, x)]\}, (f_x : A_x \rightarrow A_x^\otimes \boxplus A_x^\otimes))$ where $f_x = $ 	For objects A, B , we let $\text{merge}(A, B) = (\{$ $(0, [x_1, \dots, x_k]) \mapsto [(0, x_1), \dots, (0, x_k)],$ $(1, [y_1, \dots, y_n]) \mapsto [(1, y_1), \dots, (1, y_n)]\},$ $(\text{id}_{[\boxplus_{x \in I} A_x^\otimes]} \text{!mset}(X) + (\text{id}_{[\boxplus_{y \in I} B_y^\otimes]} \text{!mset}(Y)))$
It satisfies the following commute diagram for naturality: $\begin{array}{ccc} A & \xrightarrow{\text{bif}(A)} & F(A + A) \\ \downarrow f & & \downarrow F(f+f) \\ B & \xrightarrow{\text{bif}(B)} & F(B + B) \end{array}$	It satisfies the following commute diagram for naturality: $\begin{array}{ccc} F(A) + F(B) & \xrightarrow{\text{merge}(A, B)} & F(A + B) \\ \downarrow F(f)+F(g) & & \downarrow F(f+g) \\ F(A') + F(B') & \xrightarrow{\text{merge}(A', B')} & F(A' + B') \end{array}$

Table 4.5: Definition of bif and merge

We show the properties related to bif and merge in Lemma 4.5.3.

Lemma 4.5.3. *The following diagrams commute:*

$$\begin{array}{ccc}
A \otimes B & \xrightarrow{\text{bif}_{A \otimes B}} & F(A \otimes B + A \otimes B) \\
\downarrow \text{bif}_A \otimes \text{id} & & \uparrow F(\star) \\
F(A + A) \otimes B & \xrightarrow{t} & F((A + A) \otimes B)
\end{array}$$

$$\begin{array}{ccc}
(F A + F B) \otimes C & \xrightarrow{\text{merge} \otimes \text{id}} & F(A + B) \otimes C \\
\downarrow \star & & \downarrow t \\
F A \otimes C + F B \otimes C & & F((A + B) \otimes C) \\
\downarrow t+t & & \downarrow F(\star) \\
F(A \otimes C) + F(B \otimes C) & \xrightarrow{\text{merge}} & F(A \otimes C + B \otimes C)
\end{array}$$

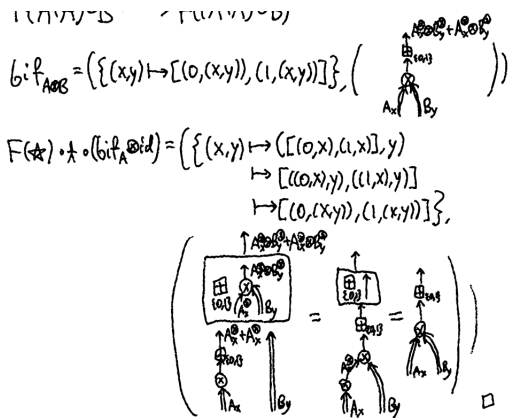
$$\begin{array}{ccc}
F^2 A + F^2 B & \xrightarrow{\text{merge}} & F(F A + F B) \\
\downarrow \mu + \mu & & \downarrow F(\text{merge}) \\
F A + F B & & F^2(A + B) \\
& \searrow \text{merge} & \downarrow \mu \\
& & F(A + B)
\end{array}$$

$$\begin{array}{ccc}
F A + F A & \xrightarrow{\text{merge}} & F(A + A) \\
& \searrow (id(F A), id(F A)) & \downarrow F(id_A, id_B) \\
& & F A
\end{array}$$

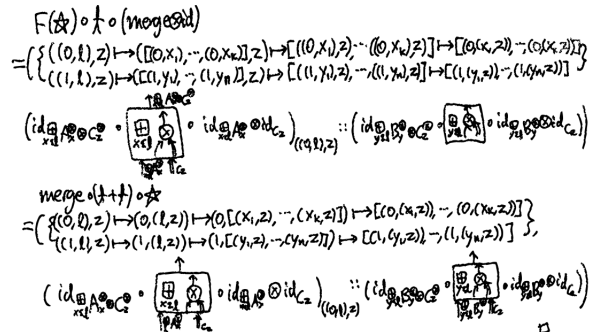
And the following equation holds.

$$\text{merge}(A, B) = (F(i_1(A, B)), F(i_2(A, B)))$$

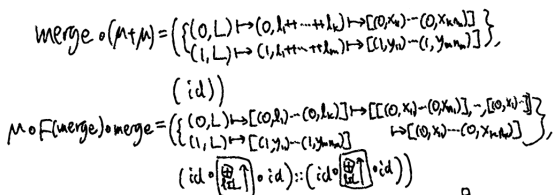
Proof. The commute diagrams can be shown as in Figure 4.28.



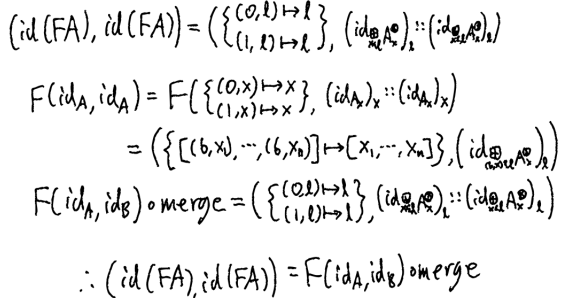
(a) Proof of the first commute diagram



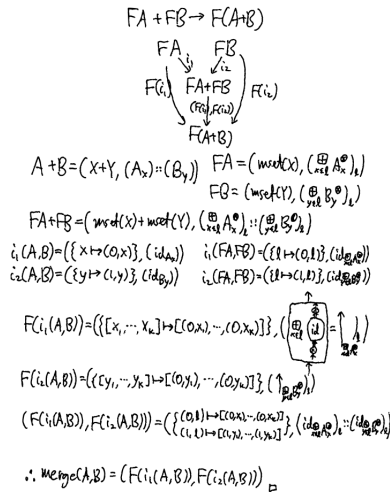
(b) Proof of the second commute diagram



(c) Proof of the third commute diagram



(d) Proof of the fourth commute diagram



(e) Proof of $\text{merge}(A, B) = (F(i_1(A, B)), F(i_2(A, B)))$

Figure 4.28: Proof of the properties of bif and merge

□

A quantum channel Q is interpreted as a morphism in the Kleisli category

$\llbracket \text{in}(Q) \rrbracket \rightarrow_{\overline{M}_F} \llbracket \text{out}(Q) \rrbracket$, where the set of interpretation of the variables $\text{in}(Q)$ and of the pattern of variables $\text{out}(Q)$ is defined as in Eq. (4.21).

$$\begin{aligned} \llbracket \text{in}(Q) \rrbracket &= (\{\emptyset\}, ([q]^{\otimes |\text{in}(Q)|})) \\ \llbracket \text{out}(Q) \rrbracket &= \begin{cases} (\{\emptyset\}, ([q]^{\otimes |V|})) & \text{if } \text{out}(Q) \text{ is a set } V \\ \llbracket o_1 \rrbracket + \llbracket o_2 \rrbracket & \text{if } \text{out}(Q) = [o_1, o_2] \end{cases} \end{aligned} \quad (4.21)$$

We assume that there is a linear ordering for the wire names of the quantum channel and that the i -th q in the list corresponds to the i -th variable of the set of variables, according to the order for the wire names.

Then the interpretation of the quantum channel is defined inductively as in Eq. (4.22).

$$\begin{aligned} \llbracket \epsilon(V) \rrbracket &= \eta(\{\emptyset\}, ([q]^{\otimes |V|})) \\ \llbracket U(\vec{V}_1) Q \rrbracket &= \llbracket Q \rrbracket \circ \llbracket U(\vec{V}_1) \rrbracket^0 \\ \llbracket \text{free } v Q \rrbracket &= \llbracket Q \rrbracket \circ \llbracket \text{free}(V) \rrbracket^0 \\ \llbracket \text{init } b v Q \rrbracket &= \llbracket Q \rrbracket \circ \llbracket \text{init}(b, v) \rrbracket^0 \\ \llbracket \text{meas } v Q_1 Q_2 \rrbracket &= \llbracket \text{in}(\text{meas } v Q_1 Q_2) \rrbracket \xrightarrow{\text{bif}} \begin{aligned} &F(\llbracket \text{in}(\text{meas } v Q_1 Q_2) \rrbracket \\ &+ \llbracket \text{in}(\text{meas } v Q_1 Q_2) \rrbracket) \end{aligned} \quad (4.22) \\ &\xrightarrow{F(\llbracket Q_1 \rrbracket \circ \llbracket \text{meas}(v, 0) \rrbracket^0 \\ &+ \llbracket Q_2 \rrbracket \circ \llbracket \text{meas}(v, 1) \rrbracket^0)} F(F \llbracket \text{out}(Q_1) \rrbracket + F \llbracket \text{out}(Q_2) \rrbracket)) \\ &\xrightarrow{F(\text{merge})} F^2(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket)) \\ &\xrightarrow{\mu} F(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket)) \end{aligned}$$

where

$$\begin{aligned} \llbracket U(V) \rrbracket &= (\{\phi \mapsto \phi\}, (\text{UV})) \\ \llbracket \text{free}(v) \rrbracket &= (\{\phi \mapsto \phi\}, (\text{free } v)) \\ \llbracket \text{init}(b, v) \rrbracket &= (\{\phi \mapsto \phi\}, (\text{init } b)) \\ &\quad (\sigma: \text{reorder the wires according to the order of the names}) \\ \llbracket \text{meas}(v, b) \rrbracket &= (\{\phi \mapsto \phi\}, (\text{meas } v, \text{init } b)) \\ &\quad : \llbracket \text{in}(Q) \rrbracket \rightarrow \llbracket \text{in}(Q) \rrbracket \end{aligned}$$

In order to interpret quantum channel constants in the type system, we need to be able to construct a morphism for the $v\text{Bind}$ relation. However, since each leaf of the branching structure depends on the typing derivation of a non-branching term, the interpretation is parameterized by these typing derivations. Formally, we represent the interpretation of $v\text{Bind}$ as $\llbracket (\tau_i)_{i:\text{leaf}} \mid v\text{Bind}(!\Delta, c, m, A) \rrbracket$

where $(\tau_i)_{i:\text{leaf}}$ denotes to the family of the typing derivation of the typing judgement at each leaf i . It is defined inductively as follows:

- for (vBind_{nb}) :

$$\frac{Q \cap \text{FV}(!\Delta) = \emptyset \quad \frac{\vdots}{! \Delta, \text{TC}_Q(Q) \vdash M : A}(\tau)}{\text{vBind}(!\Delta, Q, M, A)}(\text{vBind}_{nb})$$

We let the interpretation of the non-branching case $\llbracket \tau \mid \text{vBind}(!\Delta, Q, M, A) \rrbracket : \llbracket !\Delta \rrbracket \otimes \llbracket \text{qubit} \rrbracket^{\otimes |Q|} \rightarrow_{\overline{M}_F} \llbracket A \rrbracket$ to be the interpretation of the typing derivation of $\llbracket \tau \mid !\Delta, \text{TC}_Q(Q) \vdash M : A \rrbracket$ as in Eq (4.23).

$$\llbracket \tau \mid \text{vBind}(!\Delta, Q, M, A) \rrbracket ::= \llbracket \tau \mid !\Delta, (Q : \text{qubit}) \vdash M : A \rrbracket \quad (4.23)$$

Note that Lemma 3.2.3 implies from $! \Delta, \text{TC}_Q(Q) \vdash M : A$ that $\text{FV}(Q) \subseteq \text{FV}(M)$.

- for (vBind_b) :

$$\frac{\text{vBind}(!\Delta, c_a, m_a, A) \quad \text{vBind}(!\Delta, c_b, m_b, A)}{\text{vBind}(!\Delta, [c_a, c_b], [m_a, m_b], A)}(\text{vBind}_b)$$

We let

$$\llbracket (\tau_i) + + (\tau_j) \mid \text{vBind}(!\Delta, [c_a, c_b], [m_a, m_b], A) \rrbracket ::= f_{\text{bind}} \quad (4.24)$$

where

$$\begin{aligned} f_{\text{bind}} &= \llbracket !\Delta \rrbracket \otimes (\llbracket C_a \rrbracket + \llbracket C_b \rrbracket) \xrightarrow{\star'}_{\overline{M}} \llbracket !\Delta \rrbracket \otimes \llbracket C_a \rrbracket + \llbracket !\Delta \rrbracket \otimes \llbracket C_b \rrbracket \xrightarrow{(f_1, f_2)}_{\overline{M}_F} \llbracket A \rrbracket \\ &: \llbracket !\Delta \rrbracket \otimes \llbracket [C_a, C_b] \rrbracket = \llbracket !\Delta \rrbracket \otimes (\llbracket C_a \rrbracket + \llbracket C_b \rrbracket) \rightarrow_{\overline{M}_F} \llbracket A \rrbracket \end{aligned}$$

and

$$\begin{aligned} f_1 &= \llbracket (\tau_i) \mid \text{vBind}(!\Delta, c_a, m_a, A) \rrbracket : \llbracket !\Delta \rrbracket \otimes \llbracket C_a \rrbracket \rightarrow_{\overline{M}_F} \llbracket A \rrbracket \\ f_2 &= \llbracket (\tau_j) \mid \text{vBind}(!\Delta, c_b, m_b, A) \rrbracket : \llbracket !\Delta \rrbracket \otimes \llbracket C_b \rrbracket \rightarrow_{\overline{M}_F} \llbracket A \rrbracket \end{aligned}$$

inferrule Note that \star' refers to the left distributivity defined in Subsection 4.2.4.

4.5.3 . Interpretation of typing rules

In order to define the interpretation of typing derivation, we need an extra structure regarding the duplicable type $(!A)$, which is a parameter object in the categorical semantics $\llbracket !A \rrbracket = (p \circ b) \llbracket A \rrbracket$. This extra structure consists of the deletion and the duplication of duplicable data, which are formalized as natural transformations over the functors from the subcategory of parameter

del_{Set}	dup_{Set}
For a set X , we let $\text{del}_{\text{Set}}(X) ::= \{x \mapsto \emptyset \mid x \in X\}, (\text{id}_I)_X$ $: X \rightarrow_{\text{Set}} \{\emptyset\}$	For a set X , we let $\text{dup}_{\text{Set}}(X) ::= \{x \mapsto (x, x) \mid x \in X\}$ $: X \rightarrow_{\text{Set}} X \times X$
naturality condition: for any function $f : X \rightarrow Y$, $\begin{array}{ccc} X & \xrightarrow{\text{del}_{\text{Set}}(X)} & \{\emptyset\} \\ \downarrow f & & \downarrow \text{id}_{\{\emptyset\}} \\ Y & \xrightarrow{\text{del}_{\text{Set}}(Y)} & \{\emptyset\} \end{array}$	naturality condition: for any function $f : X \rightarrow Y$, $\begin{array}{ccc} X & \xrightarrow{\text{dup}_{\text{Set}}(X)} & X \times X \\ \downarrow f & & \downarrow f \times f \\ Y & \xrightarrow{\text{dup}_{\text{Set}}(Y)} & Y \times Y \end{array}$

Table 4.6: Definition of del_{Set} and dup_{Set} in category **Set**

objects. In other word, these natural transformation can be defined as the image of the functor $p : \text{Set} \rightarrow \overline{\overline{M}}$ from Benton's linear/non-linear category in Subsection 4.3.1 applied to the natural transformations dup_{Set} and del_{Set} which are defined in Table 4.6. We then define the following two natural transformations del and dup as in Table 4.7.

We show useful properties on del and dup in Lemma 4.5.4 which are used in the proof of soundness of the categorical semantics.

Lemma 4.5.4. *Let us let $A = (X, (I)_X)$, $B = (Y, (I)_Y)$ and $C = (Z, (I)_Z)$ be any parameter objects in the category $\overline{\overline{M}}$ and let $f : X \rightarrow Y$ and $h : X \rightarrow Z$ be any set functions from X to Y and from X to Z , respectively. Moreover, let cp be the set function defined in Eq. (4.25) and let $g : A \otimes A \otimes B \rightarrow C$ be any morphism in $\overline{\overline{M}}$ from $A \otimes A \otimes B \rightarrow C$.*

$$cp ::= \{(x, m_x) \mapsto (x, m_x \otimes m_x)\} : \sum_{x \in X} \overline{M}(I, I) \rightarrow \sum_{x \in X} \overline{M}(I, I) \otimes \overline{M}(I, I), \quad (4.25)$$

Then, the following diagrams commute,

$$\begin{array}{ccc} A & \xrightarrow{\text{dup}} & A \otimes A \\ \downarrow p(f) & & \downarrow p(f) \otimes p(f) \\ B & \xrightarrow{\text{dup}} & B \otimes B \end{array} \quad \begin{array}{ccc} A & \xrightarrow{\text{dup}} & A \otimes A \\ \downarrow \text{dup} & & \downarrow \text{id} \otimes \text{dup} \\ A \otimes A & \xrightarrow{\text{dup} \otimes \text{id}} & A \otimes A \otimes A \end{array}$$

$$\begin{array}{ccc} !A & \xrightarrow{\text{dup}} & !A \otimes !A \\ \downarrow p(cp) & & \downarrow \pi \\ !A & \xrightarrow{! \text{dup}} & !(A \otimes A) \end{array} \quad \begin{array}{ccc} A \otimes B & \xrightarrow{\text{dup}} & A \otimes B \otimes A \otimes B \\ \downarrow \text{dup} \otimes \text{dup} & \nearrow \text{id} \otimes \sigma \otimes \text{id} & \\ A \otimes A \otimes B \otimes B & & \end{array}$$

del	dup
<p>For a parameter object $A = (X, (I)_X)$, we let</p> $\begin{aligned} \text{del}(A) &::= (\{x \mapsto \emptyset\}, (\text{id}_I)_X) \\ &= p(\text{del}_{\text{Set}}(X)) \\ &: A \rightarrow_{\overline{M}} I \end{aligned}$ <p>where</p> $p(\{\emptyset\}) = (\{\emptyset\}, (I)) = I$	<p>For a parameter object $A = (X, (I)_X)$, we let</p> $\begin{aligned} \text{dup}(A) &::= (\{x \mapsto (x, x)\}, (\text{id}_I)_X) \\ &= p(\text{dup}_{\text{Set}}(X)) \\ &: A \rightarrow_{\overline{M}} A \otimes A \end{aligned}$ <p>where</p> $\begin{aligned} p(X \times X) &= (X \times X, (I)_{X \times X}) \\ &= (X \times X, (I \otimes I)_{X \times X}) \\ &= A \otimes A \end{aligned}$
<p>Naturality condition: for parameter objects A and B and morphism f in \overline{M},</p> $\begin{array}{ccc} A & \xrightarrow{\text{del}(A)} & I \\ \downarrow f & & \downarrow \text{id}_I \\ B & \xrightarrow{\text{del}(B)} & I \end{array}$	<p>Naturality condition: for parameter objects A and B and morphism f in \overline{M},</p> $\begin{array}{ccc} A & \xrightarrow{\text{dup}(A)} & A \otimes A \\ \downarrow f & & \downarrow f \otimes f \\ B & \xrightarrow{\text{dup}(B)} & B \otimes B \end{array}$

Table 4.7: Definition of del and dup

$$\begin{array}{ccc} b(A) & \xrightarrow{\text{dup}_{\text{Set}}(b(A))} & b(A) \otimes b(A) \\ \downarrow cp & & \downarrow n(A,A) \\ b(A) & \xrightarrow{b(\text{dup}(A))} & b(A \otimes A) \end{array} \quad \begin{array}{ccc} p(X) & \xrightarrow{\text{dup}(p(X))} & p(X) \otimes p(X) \\ & \searrow p(\text{dup}_{\text{Set}}(X)) & \downarrow m(X,X) \\ & & p(X \otimes X) \end{array}$$

$$\begin{array}{ccc} A \otimes B & \xrightarrow{\text{dup} \otimes \text{id}} & A \otimes A \otimes B \\ & & \downarrow \sigma \otimes \text{id} \quad \searrow g \\ & & A \otimes A \otimes B \xrightarrow{g} C \end{array} \quad \begin{array}{ccc} A & \xrightarrow{p(h)} & C \\ \downarrow p(f) & & \downarrow \text{del} \\ B & \xrightarrow{\text{del}} & I \end{array}$$

$$\begin{array}{ccc} A & & \\ \downarrow \text{dup} & \searrow \text{id} & \\ A \otimes A & \xrightarrow{\text{del} \otimes \text{id}} & A \end{array}$$

Proof. We prove the diagrams from top to bottom and from left to right.

- First diagram follows from the naturality of dup_{Set} . Explicitly, it can be shown that

$$\begin{aligned}(p(f) \otimes p(f)) \circ \text{dup} &= (\{x \mapsto (x, x) \mapsto (f(x), f(x))\}, (\text{id}_I)_X) \\ \text{dup} \circ p(f) &= (\{x \mapsto f(x) \mapsto (f(x), f(x))\}, (\text{id}_I)_X)\end{aligned}$$

- Second diagram can be shown by the following equations:

$$\begin{aligned}(\text{id} \otimes \text{dup}) \circ \text{dup} &= (\{x \mapsto (x, x) \mapsto (x, (x, x))\}, (\text{id}_I)) \\ &= (\{x \mapsto (x, x) \mapsto ((x, x), x)\}, (\text{id}_I)) \\ &= (\text{dup} \otimes \text{id}) \circ \text{dup}\end{aligned}$$

Note that the second equality is given by the fact that we let associativity in \overline{M} to be identity (Subsection 4.2.2).

- To show the third diagram, let us recall that

$$\begin{aligned}!A &= \left(\sum_{x \in X} \overline{M}(I, I), (I) \right) \\ !(A \otimes A) &= \left(\sum_{(x, x') \in X \times X} \overline{M}(I, I \otimes I), (I) \right)\end{aligned}$$

and that

$$\begin{aligned}\pi(A, B) &= (\{(x, m_x : I \rightarrow_{\overline{M}} A_x), (y, m_y : I \rightarrow_{\overline{M}} B_y)\} \mapsto ((x, y), m_x \otimes m_y), (\text{id}_I)) \\ &: (!A) \otimes (!B) \rightarrow_{\overline{M}} !(A \otimes B)\end{aligned}$$

However, since A is a parameter object, it follows that

$$\begin{aligned}\pi(A, A) &= (\{(x, m_x : I \rightarrow_{b_M} I), (y, m_y : I \rightarrow_{\overline{M}} I)\} \mapsto ((x, y), m_x \otimes m_y), (\text{id}_I)) \\ &: (!A) \otimes (!A) \rightarrow_{\overline{M}} !(A \otimes A)\end{aligned}$$

Then, the commute diagram can be shown explicitly by

$$\pi \circ \text{dup} = (\{(x, m_x) \mapsto ((x, m_x), (x, m_x)) \mapsto ((x, x), m_x \otimes m_x), (\text{id}_I))$$

and

$$!(\text{dup}) \circ p(cp) = (\{(x, m_x) \mapsto (x, m_x \otimes m_x) \mapsto ((x, x), m_x \otimes m_x), (\text{id}_I))$$

where

$$\begin{aligned}!(\text{dup}) &= (\{(x, m_x) \mapsto ((x, x), m_x), (\text{id}_I)) \\ b(\text{dup}) &= \{(x, m_x) \mapsto ((x, x), m_x)\}\end{aligned}$$

Note that $m_x \otimes m_x$ is a morphism from I to I since $I \otimes I = I$. It is necessary to be able to apply $!(\text{dup})$ to $p(cp)(!A)$.

- Fourth diagram is shown explicitly as follows

$$\begin{aligned}
& (\text{id}_A \otimes \sigma \otimes \text{id}_B) \circ (\text{dup}(A) \otimes \text{dup}(B)) \\
&= (\{(x, y) \mapsto ((x, x), (y, y)) \mapsto (x, y, x, y)\}, (\sigma_{I, I} : I \rightarrow_{\overline{M}} I)) \\
&= (\{(x, y) \mapsto ((x, y), (x, y))\}, (\sigma_{I, I} : I \rightarrow_{\overline{M}} I)) \\
&= (\{(x, y) \mapsto ((x, y), (x, y))\}, (\text{id}_I)) \\
&= \text{dup}(A \otimes B)
\end{aligned}$$

where the second equation follows from the fact that the associativity is identity and the third equation follows from the fact that $\sigma_{I, I} = \text{id}_I$ from the equation theory of diagrams in Figure 4.3. Note that we are using the fact that the associativity is identity again in order to be able to apply $(\text{id}_A \otimes \sigma \otimes \text{id}_B)$ to $(\text{dup}(A) \otimes \text{dup}(B))$.

- Fifth diagram is shown as follows:

$$\begin{aligned}
n(A, A) \circ \text{dup}_{\text{Set}}(b(A)) &= \{(x, m_x) \mapsto ((x, m_x), (x, m_x)) \mapsto ((x, x), m_x \otimes m_x)\} \\
b(\text{dup}(A)) \circ cp &= \{(x, m_x) \mapsto (x, m_x \otimes m_x) \mapsto ((x, x), m_x \otimes m_x)\}
\end{aligned}$$

where the definition of n is from the monoidal functor (b, n) from the Benton's linear/non-linear category in Subsection 4.3.1.

- Sixth diagram is shown as follows:

$$\begin{aligned}
m(X, X) \circ \text{dup}(p(X)) &= (\{x \mapsto (x, x) \mapsto (x, x)\}, (\text{id}_I)) \\
p(\text{dup}_{\text{Set}}(X)) &= (\{x \mapsto (x, x)\}, (\text{id}_I))
\end{aligned}$$

where the definition of m is from the monoidal functor (p, m) from the Benton's linear/non-linear category in Subsection 4.3.1.

- Seventh diagram is shown as follows:

$$\begin{aligned}
g \circ (\text{dup} \otimes \text{id}_B) &= (\{(x, y) \mapsto ((x, x), y) \mapsto g_0((x, x), y)\}, (g_{x, x, y})_{(x, y) \in X \times Y}) \\
g \circ (\sigma_{A, A} \otimes \text{id}_B) \circ (\text{dup} \otimes \text{id}_B) &= (\{(x, y) \mapsto ((x, x), y) \mapsto ((x, x), y) \mapsto g_0((x, x), y)\}, \\
&\quad (g_{x, x, y} \circ (\sigma_{I, I} \otimes \text{id}_{B_y}))_{(x, y) \in X \times Y}) \\
&= (\{(x, y) \mapsto ((x, x), y) \mapsto ((x, x), y) \mapsto g_0((x, x), y)\}, \\
&\quad (g_{x, x, y})_{(x, y) \in X \times Y})
\end{aligned}$$

where we let $g = (g_0, (g_{x_1, x_2, y})_{X, X, Y})$.

- Eighth diagram is shown as follows:

$$\begin{aligned}
\text{del} \circ p(f) &= (\{x \mapsto f(x) \mapsto \emptyset\}, (\text{id}_I)) \\
\text{del} \circ p(h) &= (\{x \mapsto h(x) \mapsto \emptyset\}, (\text{id}_I))
\end{aligned}$$

- Ninth diagram is shown as follows:

$$\begin{aligned} (\text{del} \otimes \text{id}_A) \circ \text{dup} &= (\{x \mapsto (x, x) \mapsto (\emptyset, x)\}, (\text{id}_I)) \\ &= (\{x \mapsto x\}, (\text{id}_I)) \end{aligned}$$

where the second equation follows from the fact that we let the natural transformation l from the monoidal structure of $\overline{\overline{M}}$ to be identity at each object, i.e., $l(A) = \text{id}_A : I \otimes A \rightarrow_{\overline{\overline{M}}} A$.

□

With the help of the extra structure on the duplicable data types defined above, we can finally define the interpretation of each typing rule as follows. To clarify the fact that the interpretation depends on the type derivation, let us denote the interpretation of type derivation π of typing derivation $! \Delta, Q \vdash m : A$ as $\llbracket \pi \mid ! \Delta, Q \vdash m : A \rrbracket$, where π represents the tree structure of typing derivation. We show useful properties on del and dup in Lemma 4.5.4 which are used in the proof of soundness of the categorical semantics.

- $\frac{}{! \Delta, (x : A) \vdash x : A} (\text{var})$:

$$\begin{aligned} &\llbracket (\text{var}) \mid ! \Delta, (x : A) \vdash x : A \rrbracket \\ &= \llbracket ! \Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{\text{del} \otimes \text{id}}_{\overline{\overline{M}}} \llbracket I \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{l_{\llbracket A \rrbracket}}_{\overline{\overline{M}}} \llbracket A \rrbracket \xrightarrow{\eta(\llbracket A \rrbracket)}_{\overline{\overline{M}}} F \llbracket A \rrbracket \end{aligned}$$

where $\eta(\llbracket A \rrbracket)$ is the unit of the lifting monad F . Note that $\llbracket I \rrbracket = I$ is the unit object from the symmetric monoidal structure of $\overline{\overline{M}}$.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\text{var}) \mid ! \Delta, (x : A) \vdash x : A \rrbracket = (\{(c, x) \mapsto (\emptyset, x) \mapsto x \mapsto [x]\}, (\begin{array}{c} \uparrow \\ \text{M}_x \\ \otimes \\ \downarrow \\ \text{A}_x \end{array})_{(c,x)})$$

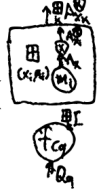
where we let $\llbracket ! \Delta \rrbracket = (C, (I)_{c \in C})$ and $\llbracket A \rrbracket = (X, (A_x)_{x \in X})$.

- $\frac{! \Delta, Q \vdash M : ! A}{! \Delta, Q \vdash M : A} (\text{d})$:


$$\begin{aligned} &\llbracket (\text{d}); \tau \mid ! \Delta, Q \vdash M : A \rrbracket \\ &= \llbracket ! \Delta \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{\llbracket \tau \mid ! \Delta, Q \vdash M : ! A \rrbracket}_{\overline{\overline{M}}} F(\llbracket ! A \rrbracket) \xrightarrow{F(\epsilon(\llbracket A \rrbracket))}_{\overline{\overline{M}}} F \llbracket A \rrbracket \end{aligned}$$

where $\epsilon(\llbracket A \rrbracket)$ is the counit of $!$ -comonad.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (d); \tau \mid !\Delta, Q \vdash M : A \rrbracket = (\{(c, q) \mapsto [(x_1, m_1), \dots, (x_k, m_k)] \mapsto [x_1, \dots, x_k]\}, \text{Diagram})_{(c, q)}$$


where we let $\llbracket !\Delta \rrbracket = (C, (I)_{c \in C})$ and $\llbracket Q \rrbracket = (Q, (Q_q)_{q \in Q})$ and

$$\llbracket \tau \mid !\Delta, Q \vdash M : !A \rrbracket = (\{(c, q) \mapsto [(x_1, m_1), \dots, (x_k, m_k)]\}, \text{Diagram})_{(c, q)}$$


$$\bullet \frac{! \Delta \vdash V : A \quad V \text{ is value}}{! \Delta \vdash V : !A} \text{(p):}$$

$$\begin{aligned} & \llbracket (p); \tau \mid !\Delta \vdash V : !A \rrbracket \\ &= (\llbracket !\Delta \rrbracket = ! [A_1] \otimes \dots \otimes ! [A_k]) \xrightarrow{\delta^{\otimes k}}_{\overline{M}} !! [A_1] \otimes \dots \otimes !! [A_k] \\ & \xrightarrow{\pi^{k-1}}_{\overline{M}} (! [A_1] \otimes \dots \otimes ! [A_k]) = ! \llbracket !\Delta \rrbracket \xrightarrow{!g}_{\overline{M}} !F [A] \xrightarrow{\phi}_{\overline{M}} F! [A] \end{aligned}$$

where δ refers to the comultiplication of $!$ -comonad and $\delta^{\otimes k} = \delta [A_1] \otimes \dots \otimes \delta [A_k]$, $\pi : !(-) \otimes !(-) \rightarrow_{\overline{M}} !(- \otimes -)$ is the strength of the monoidal comonad ($!$) and π^{k-1} represents application of π $k - 1$ times, $g = \llbracket \tau \mid !\Delta \vdash V : A \rrbracket$, and ϕ is the natural isomorphism between $!F$ and $F!$ from Subsection 4.4.3. In addition, π^{k-1} can be formally defined as follows:

$$\pi^{k-1} = \begin{cases} I \xrightarrow{\pi I}_{\overline{M}} !I & \text{if } k = 0 \\ !! [A_1] \xrightarrow{\text{id}}_{\overline{M}} !! [A_1] & \text{if } k = 1 \\ \left(\begin{array}{l} !! [A_1] \otimes (!! [A_2] \cdots !! [A_k]) \xrightarrow{\text{id} \otimes \pi^{k-2}}_{\overline{M}} \\ (!! [A_1] \otimes ! [A_2] \cdots ! [A_k]) \xrightarrow{\pi} ! (!! [A_1] \cdots !! [A_k]) \end{array} \right) & \text{otherwise} \end{cases} \quad (4.26)$$

Note that if $k = 0$, then $!\Delta = I$ and that the tensor product is associative.

Explicitly, the interpretation of the typing derivation is represented as

follows:

$$\llbracket (\text{p}); \tau \mid !\Delta \vdash V : !A \rrbracket =$$

$$\left\{ \begin{array}{l} I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{!g} \overline{M} !F \llbracket A \rrbracket \xrightarrow{\phi} \overline{M} F \llbracket A \rrbracket \\ = \left(\left\{ \begin{array}{l} \emptyset \mapsto (\emptyset, \text{id}_I) \mapsto ([x_1, \dots, x_l], g_\emptyset) \\ \mapsto [(x_1, \textcircled{1}), \dots, (x_l, \textcircled{1})] \end{array} \right\}, (\text{id}_I)_{\{\emptyset\}} \right) \\ \left(\begin{array}{l} \llbracket !\Delta \rrbracket \xrightarrow{\delta^\infty} \overline{M} !! \llbracket A_1 \rrbracket \otimes \dots \otimes !! \llbracket A_k \rrbracket \\ \xrightarrow{\pi^*} \overline{M} ! \llbracket \Delta \rrbracket \xrightarrow{!g} \overline{M} !F \llbracket A \rrbracket \xrightarrow{\phi} \overline{M} F \llbracket A \rrbracket \end{array} \right) \\ = \left(\left\{ \begin{array}{l} ((x^1, A^1), \dots, (x^k, A^k)) \\ \mapsto (((x^1, A^1), \text{id}_I), \dots, ((x^k, A^k), \text{id}_I)) \\ \mapsto (((x^1, A^1), \dots, (x^k, A^k)), \text{id}_I) \\ \mapsto ([x_1, \dots, x_l], \textcircled{\uparrow A x_i} \textcircled{1})_c \\ \mapsto [(x_1, \textcircled{1}), \dots, (x_l, \textcircled{1})] \end{array} \right\}, (\text{id}_I)_C \right) \end{array} \right. , \text{ if } !\Delta = \emptyset$$

$$\left. \begin{array}{l} \left(\begin{array}{l} ((x^1, A^1), \dots, (x^k, A^k)) \\ \mapsto (((x^1, A^1), \text{id}_I), \dots, ((x^k, A^k), \text{id}_I)) \\ \mapsto (((x^1, A^1), \dots, (x^k, A^k)), \text{id}_I) \\ \mapsto ([x_1, \dots, x_l], \textcircled{\uparrow A x_i} \textcircled{1})_c \\ \mapsto [(x_1, \textcircled{1}), \dots, (x_l, \textcircled{1})] \end{array} \right) \end{array} \right) , \text{ if } !\Delta \neq \emptyset$$

where we let $\llbracket !\Delta \rrbracket = (C, (I)_{c \in C})$, $\llbracket A_i \rrbracket = (X_i, (A_{i,x_i})_{x_i \in X_i})$, $! \llbracket A_i \rrbracket = (\sum_{x_i \in X_i} \overline{M}(I, A_{i,x_i}), (I))$, and

$$\llbracket \tau \mid !\Delta \vdash V : A \rrbracket = (g, (g_c)_c) = (\{c \mapsto [x_1, \dots, x_l]\}, \textcircled{\uparrow A x_i} \textcircled{1})_c$$

- $\overline{!\Delta \vdash * : I}$ (I):

$$\llbracket (I) \mid !\Delta \vdash * : I \rrbracket = \llbracket !\Delta \rrbracket \xrightarrow{\text{del}} \overline{M} \llbracket I \rrbracket \xrightarrow{\eta(\llbracket I \rrbracket)} \overline{M} F \llbracket I \rrbracket$$

where $\eta(\llbracket I \rrbracket)$ is the unit of the lifting monad F .

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (I) \mid !\Delta \vdash * : I \rrbracket = (\{c \mapsto \emptyset \mapsto [\emptyset] \mid c \in C\}, (\text{id}_I)_C)$$

where we let $\llbracket !\Delta \rrbracket = (C, (I)_C)$.

- $\overline{!\Delta \vdash \text{tt} : \text{bool}}$ (tt):

$$\llbracket (\text{tt}) \mid !\Delta \vdash \text{tt} : \text{bool} \rrbracket = \llbracket !\Delta \rrbracket \xrightarrow{\text{del}} \overline{M} \llbracket I \rrbracket \xrightarrow{f_{\text{tt}}} \overline{M} \llbracket \text{bool} \rrbracket \xrightarrow{\eta(\llbracket \text{bool} \rrbracket)} \overline{M} F \llbracket \text{bool} \rrbracket$$

where $f_{\text{tt}} = (\{\emptyset \mapsto \text{tt}\}, (\text{id}_I))$ and $\text{tt} = (0, \emptyset)$ are defined in Eq. (4.18), and $\eta(\llbracket \text{bool} \rrbracket)$ is the unit of the lifting monad F .

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\text{tt}) \mid !\Delta \vdash \text{tt} : \text{bool} \rrbracket = (\{c \mapsto \emptyset \mapsto \text{tt} \mapsto [\text{tt}] \mid c \in C\}, (\text{id}_I)_C)$$

where we let $\llbracket !\Delta \rrbracket = (C, (I)_C)$.

- $\frac{}{!\Delta \vdash \text{ff} : \text{bool}}(\text{ff})$:

$$\llbracket (\text{ff}) \mid !\Delta \vdash \text{ff} : \text{bool} \rrbracket = \llbracket !\Delta \rrbracket \xrightarrow{\text{del}}_{\overline{M}} \llbracket I \rrbracket \xrightarrow{f_{\text{ff}}}_{\overline{M}} \llbracket \text{bool} \rrbracket \xrightarrow{\eta(\llbracket \text{bool} \rrbracket)}_{\overline{M}} F \llbracket \text{bool} \rrbracket$$

where $f_{\text{ff}} = (\{\emptyset \mapsto \text{ff}\}, (\text{id}_I))$ and $\text{ff} = (1, \emptyset)$ are defined in Eq. (4.18), and $\eta(\llbracket \text{bool} \rrbracket)$ is the unit of the lifting monad F .

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\text{ff}) \mid !\Delta \vdash \text{ff} : \text{bool} \rrbracket = (\{c \mapsto \emptyset \mapsto \text{ff} \mapsto [\text{ff}] \mid c \in C\}, (\text{id}_I)_C)$$

where we let $\llbracket !\Delta \rrbracket = (C, (I)_C)$.

- $\frac{!\Delta, Q, (x : A_a) \vdash M : A_b}{!\Delta, Q \vdash \lambda x.M : A_a \multimap A_b}(\multimap_I)$:

$$\begin{aligned} \llbracket (\multimap_I); \tau \mid !\Delta, Q \vdash \lambda x.M : A_a \multimap A_b \rrbracket = \\ \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{\eta_{\llbracket A_a \rrbracket}(\llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket)}}_{\overline{M}} \llbracket A_a \rrbracket \multimap (\llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \otimes \llbracket A_a \rrbracket) \\ \xrightarrow{(\llbracket A_a \rrbracket \multimap -)(f)}_{\overline{M}} \llbracket A_a \rrbracket \multimap F \llbracket A_b \rrbracket \xrightarrow{\eta(\llbracket A_a \multimap A_b \rrbracket)}}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket \end{aligned}$$

where $\eta_{\llbracket A_a \rrbracket}(\llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket)$ refers to the natural transformation $\eta : \mathbf{1}_{\overline{M}} \rightarrow (\llbracket A_a \rrbracket \multimap -) \circ (- \otimes \llbracket A_a \rrbracket)$ and $(\llbracket A_a \rrbracket \multimap -)$ the functor from Subsection 4.2.2 on the internal hom of \overline{M} , $f = \llbracket \tau \mid !\Delta, Q, (x : A_a) \vdash M : A_b \rrbracket : \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \otimes \llbracket A_a \rrbracket \rightarrow_{\overline{M}} F \llbracket A_b \rrbracket$, and $\eta(\llbracket A_a \multimap A_b \rrbracket)$ is the unit of the lifting monad F .

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\multimap_I); \tau \mid !\Delta, Q \vdash \lambda x.M : A_a \multimap A_b \rrbracket = \left(\left\{ \begin{array}{l} (c, q) \mapsto \{x_a \mapsto ((c, q), x_a)\} \\ \mapsto \{x_a \mapsto f_0((c, q), x_a)\} \\ \mapsto \{x_a \mapsto f_0((c, q), x_a)\} \end{array} \right\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \\ \text{Diagram 3} \end{array} \right) \right)_{(c, q) \in C \times Q}$$

where we let $[\![\Delta]\!] = (C, (I)_C)$, $[\![Q]\!] = (Q, (Q_q)_{q \in Q})$, and $f = (f_0, (f_{(c,q,x_a)}))$. Note that $(([\![\Delta]\!] \otimes [\![Q]\!]) \otimes [\![A_a]\!]) = [\![\Delta]\!] \otimes (([\![Q]\!] \otimes [\![A_a]\!]))$ is derived from the fact that the associativity α is identity morphism.

$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \multimap A_b \quad ! \Delta, Q_b \vdash M_b : A_a}{! \Delta, Q_a, Q_b \vdash M_a M_b : A_b} ({}_{\multimap} E):$$

$$\begin{aligned} & [({}_{\multimap} E); (\tau_1, \tau_2) \mid ! \Delta, Q_a, Q_b \vdash M_a M_b : A_b] = \\ & \frac{[\![\Delta]\!] \otimes [\![Q_a]\!] \otimes [\![Q_b]\!] \xrightarrow{\text{dup} \otimes \text{id}_{[\![Q_a]\!] \otimes [\![Q_b]\!]}} \overline{M} [\![\Delta]\!] \otimes [\![\Delta]\!] \otimes [\![Q_a]\!] \otimes [\![Q_b]\!]}{\text{id}_{[\![\Delta]\!] \otimes [\![Q_b]\!] \otimes \sigma \otimes \text{id}_{[\![Q_a]\!]}} \overline{M} [\![\Delta]\!] \otimes [\![Q_a]\!] \otimes [\![\Delta]\!] \otimes [\![Q_b]\!] \xrightarrow{f \otimes g} \overline{M} F[A_a \multimap A_b] \otimes F[A_a]} \\ & \frac{\psi_{[A_a \multimap A_b], [A_a]} \overline{M} (F([A_a \multimap A_b] \otimes [A_a]) = F([\![A_a]\!] \multimap_{\overline{M}_F} [\![A_b]\!] \otimes [A_a]))}{F(\text{eval}_{[A_a], F[A_b]}) \overline{M} F^2[A_b] \xrightarrow{\mu} \overline{M} F[A_b]} \end{aligned}$$

where σ is the commutativity natural transformation of \overline{M} , $f = [\tau_1 \mid ! \Delta, Q_a \vdash M_a : A_a \multimap A_b] : [\![\Delta]\!] \otimes [\![Q_a]\!] \rightarrow \overline{M} F[A_a \multimap A_b]$ and $g = [\tau_2 \mid ! \Delta, Q_b \vdash M_b : A_a] : [\![\Delta]\!] \otimes [\![Q_b]\!] \rightarrow \overline{M} F[A_a]$, $\psi_{A,B} : FA \otimes FB \rightarrow \overline{M} F(A \otimes B)$ is the tensorial strength of the branching computation monad F (Subsection 4.4.1), $[A_a \multimap A_b] = [A_a] \multimap_{\overline{M}_F} [A_b]$ by definition, $\text{eval}_{B,A} = \epsilon(A) : (B \multimap A) \otimes B \rightarrow \overline{M} A$ refers to the evaluation defined in Subsection 4.2.2, and $\mu : F^2 \rightarrow F$ is the multiplication of branching monad from Subsection 4.4.1.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$[({}_{\multimap} E); (\tau_1, \tau_2) \mid ! \Delta, Q_a, Q_b \vdash M_a M_b : A_b] = \left(\left(\begin{array}{l} (c, q^a, q^b) \\ \mapsto (c, c, q^a, q^b) \\ \mapsto (c, q^a, c, q^b) \\ \mapsto ([h_1, \dots, h_k], [x_1^a, \dots, x_l^a]) \\ \mapsto [(h_1, x_1^a), \dots, (h_1, x_l^a), \dots, (h_k, x_1^a), \dots, (h_k, x_l^a)] \\ \mapsto [h_1(x_1^a), \dots, h_1(x_l^a), \dots, h_k(x_1^a), \dots, h_k(x_l^a)] \\ \mapsto h_1(x_1^a) + \dots + h_k(x_l^a) \end{array} \right) \right), \left(\begin{array}{c} \text{Diagram with nodes } A_i^a, A_i^b, A_i^c \text{ and arrows} \\ \text{Diagram with nodes } A_i^a, A_i^b, A_i^c \text{ and arrows} \\ \text{Diagram with nodes } A_i^a, A_i^b, A_i^c \text{ and arrows} \end{array} \right) (c, q^a, q^b)$$

where we let $[\![\Delta]\!] = (C, (I)_C)$, $[\![Q_a]\!] = (Q^a, (Q_{q^a}^a)_{q^a \in Q^a})$, $[\![Q_b]\!] = (Q^b, (Q_{q^b}^b)_{q^b \in Q^b})$, and

$$\begin{aligned} f &= (\{(c, q^a) \mapsto [h_1, \dots, h_k]\}, (f_{c,q^a})_{(c,q^a) \in C \times Q^a}) \\ g &= (\{(c, q^b) \mapsto [x_1^a, \dots, x_l^a]\}, (g_{c,q^b})_{(c,q^b) \in C \times Q^b}) \end{aligned}$$

$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \quad ! \Delta, Q_b \vdash M_b : A_b}{! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b} (\otimes_I):$$

$$\begin{aligned} & \llbracket (\otimes_I); (\tau_a, \tau_b) \mid ! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b \rrbracket = \\ & \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup}^{\otimes \text{id}}_{\llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket}} \overline{\overline{\llbracket ! \Delta \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket}}} \\ & \xrightarrow{\text{id}_{\llbracket ! \Delta \rrbracket} \otimes \sigma \otimes \text{id}_{\llbracket Q_b \rrbracket}} \overline{\overline{\llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket}}} \xrightarrow{f \otimes g} \overline{\overline{F \llbracket A_a \rrbracket \otimes F \llbracket A_b \rrbracket}}} \\ & \xrightarrow{\psi_{\llbracket A_a \rrbracket, \llbracket A_b \rrbracket}}} \overline{\overline{F(\llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket)}} = \overline{\overline{F \llbracket A_a \otimes A_b \rrbracket}}} \end{aligned}$$

where σ is the commutativity natural transformation of $\overline{\overline{M}}$,
 $f = \llbracket \tau_a \mid ! \Delta, Q_a \vdash M_a : A_a \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \rightarrow \overline{\overline{F \llbracket A_a \rrbracket}}$ and
 $g = \llbracket \tau_b \mid ! \Delta, Q_b \vdash M_b : A_b \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \rightarrow \overline{\overline{F \llbracket A_b \rrbracket}}$, $\psi_{A,B} : FA \otimes FB \rightarrow \overline{\overline{F(A \otimes B)}}$ is the tensorial strength of the branching computation monad F (Subsection 4.4.1), and $\llbracket A_a \otimes A_b \rrbracket = \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket$ by definition.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\otimes_I); (\tau_a, \tau_b) \mid ! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b \rrbracket = \left(\left(\begin{array}{l} (c, q^a, q^b) \\ \mapsto (c, c, q^a, q^b) \\ \mapsto (c, q^a, c, q^b) \\ \mapsto ([x_1^a, \dots, x_k^a], [x_1^b, \dots, x_l^b]) \\ \mapsto \left[\begin{array}{l} (x_1^a, x_1^b), \dots, (x_1^a, x_l^b), \dots, \\ (x_k^a, x_1^b), \dots, (x_k^a, x_l^b) \end{array} \right] \end{array} \right) \right), \left(\begin{array}{c} \text{Diagram} \\ (c, q^a, q^b) \end{array} \right)$$

where we let $\llbracket ! \Delta \rrbracket = (C, (I)_C)$, $\llbracket Q_a \rrbracket = (Q^a, (Q_{q^a}^a)_{q^a \in Q^a})$, $\llbracket Q_b \rrbracket = (Q^b, (Q_{q^b}^b)_{q^b \in Q^b})$, and

$$\begin{aligned} f &= (\{(c, q^a) \mapsto [x_1^a, \dots, x_k^a]\}, (f_{c, q^a})_{(c, q^a) \in C \times Q^a}) \\ g &= (\{(c, q^b) \mapsto [x_1^b, \dots, x_l^b]\}, (g_{c, q^b})_{(c, q^b) \in C \times Q^b}) \end{aligned}$$

$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \otimes A_b \quad ! \Delta, Q_b, (x : A_a), (y : A_b) \vdash M_b : A}{! \Delta, Q_a, Q_b \vdash \text{let } \langle x, y \rangle = M_a \text{ in } M_b : A} (\otimes_E):$$

$$\begin{aligned} & \llbracket (\otimes_E); (\tau_1, \tau_2) \mid ! \Delta, Q_a, Q_b \vdash \text{let } \langle x, y \rangle = M_a \text{ in } M_b : A \rrbracket = \\ & \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup}^{\otimes \text{id}}_{\llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket}} \overline{\overline{M}} \llbracket ! \Delta \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \\ & \xrightarrow{\text{id}^{\llbracket ! \Delta \rrbracket \otimes \sigma \otimes \text{id}}_{\llbracket Q_b \rrbracket}} \overline{\overline{M}} \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \\ & \xrightarrow{f^{\otimes \text{id}}_{\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket}} \overline{\overline{M}} F \llbracket A_a \otimes A_b \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \\ & \xrightarrow{t_{\llbracket A_a \otimes A_b \rrbracket, \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket}}} \overline{\overline{M}} F(\llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket) \\ & \xrightarrow{F(\sigma)} \overline{\overline{M}} F(\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \otimes \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket) \xrightarrow{F(g)} \overline{\overline{M}} F(F \llbracket A \rrbracket) \xrightarrow{\mu} \overline{\overline{M}} F \llbracket A \rrbracket \end{aligned}$$

where σ is the commutativity natural transformation of $\overline{\overline{M}}$,
 $f = \llbracket \tau_1 \mid ! \Delta, Q_a \vdash M_a : A_a \otimes A_b \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \rightarrow \overline{\overline{M}} F \llbracket A_a \otimes A_b \rrbracket$, $t_{A,B} : F A \otimes B \rightarrow \overline{\overline{M}} F(A \otimes B)$ is the tensorial strength of the branching computation monad F from Subsection 4.4.1, $\llbracket A_a \otimes A_b \rrbracket = \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket$ by definition, $g = \llbracket \tau_2 \mid ! \Delta, Q_b, (x : A_1), (y : A_2) \vdash M_b : A \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \otimes \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket \rightarrow \overline{\overline{M}} F \llbracket A \rrbracket$, and $\mu : F^2 \rightarrow F$ is the multiplication of branching monad from Subsection 4.4.1.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\otimes_E); (\tau_1, \tau_2) \mid ! \Delta, Q_a, Q_b \vdash \text{let } \langle x, y \rangle = M_a \text{ in } M_b : A \rrbracket = \left(\left(\begin{array}{l} (c, q^a, q^b) \\ \mapsto (c, c, q^a, q^b) \\ \mapsto (c, q^a, c, q^b) \\ \mapsto ((x_1^a, x_1^b), \dots, (x_k^a, x_k^b)), c, q^b \\ \mapsto [(x_1^a, x_1^b, c, q^b), \dots, (x_k^a, x_k^b, c, q^b)] \\ \mapsto [(c, q^b, x_1^a, x_1^b), \dots, (c, q^b, x_k^a, x_k^b)] \\ \mapsto [x_1^1, \dots, x_{l_1}^1], \dots, \\ [x_1^k, \dots, x_{l_k}^k] \\ \mapsto [x_1^1, \dots, x_{l_1}^1, \dots, x_1^k, \dots, x_{l_k}^k] \end{array} \right), \left(\begin{array}{c} \text{Diagram illustrating the interpretation of the typing derivation, showing the flow of data and the application of the monad F. The diagram consists of several boxes and arrows, with labels like A_a, A_b, Q_a, Q_b, and F. The diagram is enclosed in a large right-facing curly bracket with the label (c, q^a, q^b) at the bottom right. The diagram shows a sequence of operations: a box for A_a and A_b, a box for Q_a and Q_b, and a box for F. Arrows indicate the flow of data between these boxes, with some arrows labeled with variables like x_1^a, x_1^b, etc. The diagram is a complex representation of the typing derivation's interpretation.$$

where we let $\llbracket ! \Delta \rrbracket = (C, (I)_C)$, $\llbracket Q_a \rrbracket = (Q^a, (Q_{q^a}^a)_{q^a \in Q^a})$, $\llbracket Q_b \rrbracket = (Q^b, (Q_{q^b}^b)_{q^b \in Q^b})$, $\llbracket A_a \rrbracket = (X^a, (A_{x^a}^a)_{x^a \in X^a})$, $\llbracket A_b \rrbracket = (X^b, (A_{x^b}^b)_{x^b \in X^b})$, and

$$\begin{aligned} f &= \{(c, q^a) \mapsto [(x_1^a, x_1^b), \dots, (x_k^a, x_k^b)]\}, (f_{c, q^a})_{(c, q^a) \in C \times Q^a} \\ g &= \{(c, q^b, x^a, x^b) \mapsto [x_1, \dots, x_l]\}, (g_{c, q^b, x^a, x^b})_{(c, q^b, x^a, x^b) \in C \times Q^b \times X^a \times X^b} \end{aligned}$$

$$\bullet \frac{! \Delta, Q_a \vdash M : \text{bool} \quad ! \Delta, Q_b \vdash M_a : A \quad ! \Delta, Q_b \vdash M_b : A}{! \Delta, Q_a, Q_b \vdash \text{if } M \text{ then } M_a \text{ else } M_b : A} (\text{if}):$$

$$\llbracket (\text{if}); (\tau_M, \tau_a, \tau_b) \mid ! \Delta, Q_a, Q_b \vdash \text{if } M \text{ then } M_a \text{ else } M_b : A \rrbracket =$$

$$\begin{aligned} & \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup} \otimes \text{id}_{\llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket}}} \overline{\overline{M}} \llbracket ! \Delta \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \\ & \xrightarrow{\text{id}_{\llbracket ! \Delta \rrbracket} \otimes \sigma \otimes \text{id}_{\llbracket Q_b \rrbracket}}} \overline{\overline{M}} \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \\ & \xrightarrow{f \otimes \text{id}_{\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket}}} \overline{\overline{M}} F[\text{bool}] \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \\ & \xrightarrow{t_{\llbracket \text{bool} \rrbracket, \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket}}} \overline{\overline{M}} (F(\llbracket \text{bool} \rrbracket) \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket) = F((\llbracket I \rrbracket + \llbracket I \rrbracket) \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket)) \\ & \xrightarrow{F(\star)} \overline{\overline{M}} F((\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket) + (\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket)) \\ & \xrightarrow{F(g_a, g_b)} \overline{\overline{M}} F(F \llbracket A \rrbracket) \xrightarrow{\mu} \overline{\overline{M}} F \llbracket A \rrbracket \end{aligned}$$

where σ is the commutativity natural transformation of $\overline{\overline{M}}$;

$f = \llbracket \tau_M \mid ! \Delta, Q_a \vdash M : \text{bool} \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \rightarrow \overline{\overline{M}} F[\text{bool}]$; $t_{A,B} : FA \otimes B \rightarrow \overline{\overline{M}} F(A \otimes B)$ is the tensorial strength of the branching computation monad F from Subsection 4.4.1; $\llbracket \text{bool} \rrbracket = (\{\text{tt}, \text{ff}\}, (I, I)) = \llbracket I \rrbracket + \llbracket I \rrbracket$; $\star(A, B) : (A + B) \otimes C \rightarrow (A \otimes C) + (B \otimes C)$ is the natural transformation for distributivity of \otimes over $+$ from Subsection 4.2.4; $\llbracket I \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket = \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket$ is derived from the fact that we let the natural transformation l_A from the monoidal structure of $\overline{\overline{M}}$ to be the identity morphism for each object A ; $g_a = \llbracket \tau_a \mid ! \Delta, Q_b \vdash M_a : A \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \rightarrow \overline{\overline{M}} F \llbracket A \rrbracket$, $g_b = \llbracket \tau_b \mid ! \Delta, Q_b \vdash M_b : A \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \rightarrow \overline{\overline{M}} F \llbracket A \rrbracket$, and $(g_a, g_b) : (\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket) + (\llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket) \rightarrow \overline{\overline{M}} F \llbracket A \rrbracket$ is defined from the coproduct of $\overline{\overline{M}}$ in Subsection 4.2.3; and $\mu : F^2 \rightarrow F$ is the multiplication of branching monad from Subsection 4.4.1.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\text{if}); (\tau_M, \tau_a, \tau_b) \mid ! \Delta, Q_a, Q_b \vdash \text{if } M \text{ then } M_a \text{ else } M_b : A \rrbracket =$$

$$\left(\left(\begin{array}{l} (c, q^a, q^b) \\ \mapsto (c, c, q^a, q^b) \\ \mapsto (c, q^a, c, q^b) \\ \mapsto ([b_1, \dots, b_k], c, q^b) \\ \mapsto [(b_1, c, q^b), \dots, (b_k, c, q^b)] \\ \mapsto [(01_1, (\emptyset, c, q^b)), \dots, (01_k, (\emptyset, c, q^b))] \\ \mapsto [(01_1, (c, q^b)), \dots, (01_k, (c, q^b))] \\ \mapsto [g_{01_1}(c, q^b), \dots, g_{01_k}(c, q^b)] \\ \mapsto g_{01_1}(c, q^b) + \dots + g_{01_k}(c, q^b) \end{array} \right) \right), \left(\begin{array}{c} \text{Hand-drawn diagram showing the semantic interpretation of the if-then-else expression with nodes for monads and objects.} \\ \text{The diagram includes boxes for } \text{Mat}_k, \text{ and } \text{Mat}_k, \text{ with arrows representing morphisms } f, g, \text{ and } \mu. \\ \text{Labels include } c, q^a, q^b, \text{ and } A_{X_{i^k}}. \\ \text{The bottom part is labeled } (c, q^a, q^b). \end{array} \right)$$

where we let $\llbracket !\Delta \rrbracket = (C, (I)_C)$, $\llbracket Q_a \rrbracket = (Q^a, (Q_{q^a}^a)_{q^a \in Q^a})$, $\llbracket Q_b \rrbracket = (Q^b, (Q_{q^b}^b)_{q^b \in Q^b})$, $b_i = (01_i, \emptyset) \in \{\text{tt}, \text{ff}\}$, and

$$\begin{aligned} f &= (\{(c, q^a) \mapsto [b_1, \dots, b_k]\}, (f_{c, q^a})_{(c, q^a) \in C \times Q^a}) \\ g_a &= \llbracket !\Delta, Q_b \vdash M_a : A \rrbracket = (g_0, (g_{0, c, q^b})_{(c, q^b) \in C \times Q^b}) \\ g_b &= \llbracket !\Delta, Q_b \vdash M_b : A \rrbracket = (g_1, (g_{1, c, q^b})_{(c, q^b) \in C \times Q^b}) \end{aligned}$$

$$\bullet \frac{}{! \Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A)} (\text{box}):$$

$$\begin{aligned} &\llbracket (\text{box}) \mid ! \Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket = \\ &\llbracket !\Delta \rrbracket \xrightarrow{\text{del}}_{\overline{M}} I \xrightarrow{\eta_{\llbracket P \multimap A \rrbracket}(I)}}_{\overline{M}} ! \llbracket P \multimap A \rrbracket \multimap (I \otimes ! \llbracket P \multimap A \rrbracket) \\ &\xrightarrow{(! \llbracket P \multimap A \rrbracket \multimap -)(\eta \circ \text{box})}_{\overline{M}} ! \llbracket P \multimap A \rrbracket \multimap F(! \llbracket \text{QChan}(P, A) \rrbracket) \\ &\xrightarrow{\eta}_{\overline{M}} F \llbracket !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket \end{aligned}$$

where $\eta_{\llbracket P \multimap A \rrbracket}(I)$ refers to the natural transformation $\eta : \mathbf{1}_{\overline{M}} \rightarrow (! \llbracket P \multimap A \rrbracket \multimap -) \circ (- \otimes ! \llbracket P \multimap A \rrbracket)$ and $(! \llbracket P \multimap A \rrbracket \multimap -)$ refers to the functor from Subsection 4.2.2; $I \otimes ! \llbracket P \multimap A \rrbracket = ! \llbracket P \multimap A \rrbracket$ is derived from the fact that we let the natural transformation $l_A : I \otimes A \rightarrow A$ from the monoidal structure of \overline{M} to be identity; $\eta \circ \text{box}$ is the composition of the morphism $\text{box} : (! \llbracket A \rrbracket \multimap_{\overline{M}} F \llbracket B \rrbracket) = (p \circ b)(\llbracket A \rrbracket \multimap_{\overline{M}} F \llbracket B \rrbracket) \rightarrow_{\overline{M}} (! \llbracket \text{QChan}(A, B) \rrbracket) = (p \circ b \circ p)(\overline{M}(\llbracket A \rrbracket, F \llbracket B \rrbracket))$ defined in Subsection 4.5.1 and the unit $\eta(A) : A \rightarrow F(A)$ is the unit of the lifting monad F ; $! \llbracket P \multimap A \rrbracket \multimap F(! \llbracket \text{QChan}(P, A) \rrbracket) = \llbracket !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket$; and the second η also refers to the unit of the lifting monad F .

Explicitly, the interpretation of the typing derivation is represented as

follows:

$$\llbracket (\text{box}) \mid !\Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket =$$

$$\left(\begin{array}{l} c \mapsto \emptyset \\ \mapsto \{(f, m_f) \mapsto (\emptyset, (f, m_f))\} \\ \mapsto \left\{ \begin{array}{l} (f, m_f) \mapsto (\emptyset, (f, m_f)) \mapsto (f, m_f) \\ \mapsto [((f, (P_p)_{p \in P}), \text{id}_I)] \end{array} \right\} \\ \mapsto \left\{ \begin{array}{l} (f, m_f) \mapsto (\emptyset, (f, m_f)) \mapsto (f, m_f) \\ \mapsto [((f, (P_p)_{p \in P}), \text{id}_I)] \end{array} \right\} \end{array} \right), \left(\begin{array}{l} \uparrow \text{!}A^p (I \multimap I)^\otimes \\ \uparrow (I \multimap I)^\otimes \\ \uparrow \text{!}A^p (I \multimap I)^\otimes \\ \uparrow (I \multimap I)^\otimes \\ \uparrow \text{!}A^p (I \multimap I)^\otimes \\ \uparrow (I \multimap I)^\otimes \\ \uparrow \text{!}A^p (I \multimap I)^\otimes \\ \uparrow (I \multimap I)^\otimes \\ \uparrow \text{!}A^p (I \multimap I)^\otimes \\ \uparrow (I \multimap I)^\otimes \end{array} \right) \quad (c)$$

where we let $\llbracket !\Delta \rrbracket = (C, (I)_C)$, $\llbracket Q_a \rrbracket = (Q^a, (Q_{q^a}^a)_{q^a \in Q^a})$, $\llbracket Q_b \rrbracket = (Q^b, (Q_{q^b}^b)_{q^b \in Q^b})$, $b_i = (01_i, \emptyset) \in \{\text{tt}, \text{ff}\}$. Moreover, note that we represent the interpretation of each type explicitly as follows.

$$\begin{aligned} \llbracket P \rrbracket &= (P, (P_p)_{p \in P}) \\ \llbracket A \rrbracket &= (X, (A_x)_{x \in X}) \\ \llbracket P \multimap A \rrbracket &= (P \rightarrow \text{mset}(X), (\boxplus_{p \in P} P_p^\perp \otimes (\boxplus_{x \in f(p)} A_x^\otimes))_{f: P \rightarrow \text{mset}(X)}) \\ !\llbracket P \multimap A \rrbracket &= \left(\sum_{f: P \rightarrow \text{mset}(X)} \overline{M}(I, \boxplus_{p \in P} P_p^\perp \otimes (\boxplus_{x \in f(p)} A_x^\otimes)), (I) \right) \\ &= (!A^p, (I)) \\ !\llbracket P \multimap A \rrbracket \multimap (I \otimes !\llbracket P \multimap A \rrbracket) &= (!A^p \rightarrow \{\emptyset\} \times !A^p, (\boxplus_{x: !A^p} (I \multimap (I \otimes I))^\otimes)_{f: !A^p \rightarrow \{\emptyset\} \times !A^p}) \end{aligned}$$

by letting

$$!A^p = \sum_{f: P \rightarrow \text{mset}(X)} \overline{M}(I, \boxplus_{p \in P} P_p^\perp \otimes (\boxplus_{x \in f(p)} A_x^\otimes))$$

and

$$(f : P \rightarrow \text{mset}(X), m_f : I \rightarrow_{\overline{M}} \boxplus_{p \in P} P_p^\perp \otimes (\boxplus_{x \in f(p)} A_x^\otimes)) : !A^p$$

- $\overline{! \Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A)}^{\text{(unbox)}}$:

$$\begin{aligned}
& \llbracket (\text{unbox}) \mid ! \Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket = \\
& \llbracket ! \Delta \rrbracket \xrightarrow{\text{del}}_{\overline{M}} I \xrightarrow{\eta_{\llbracket \text{QChan}(P, A) \rrbracket}(I)}}_{\overline{M}} \llbracket \text{QChan}(P, A) \rrbracket \multimap (I \otimes \llbracket \text{QChan}(P, A) \rrbracket) \\
& \xrightarrow{(\llbracket \text{QChan}(P, A) \rrbracket \multimap -)(\eta \circ \text{unbox})}}_{\overline{M}} \llbracket \text{QChan}(P, A) \rrbracket \multimap F \llbracket P \multimap A \rrbracket \\
& \xrightarrow{\eta}_{\overline{M}} F \llbracket \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket
\end{aligned}$$

where $\eta_{\llbracket \text{QChan}(P, A) \rrbracket}(I)$ refers to the natural transformation $\eta : \mathbf{1}_{\overline{M}} \rightarrow (\llbracket \text{QChan}(P, A) \rrbracket \multimap -) \circ (- \otimes \llbracket \text{QChan}(P, A) \rrbracket)$ and $(\llbracket \text{QChan}(P, A) \rrbracket \multimap -)$ refers to the functor from Subsection 4.2.2; $I \otimes \llbracket \text{QChan}(P, A) \rrbracket = \llbracket \text{QChan}(P, A) \rrbracket$ is derived from the fact that we let the natural transformation $l_A : I \otimes A \rightarrow A$ from the monoidal structure of \overline{M} to be identity; $\eta \circ \text{unbox}$ is the composition of the morphism $\text{unbox} : (\llbracket \text{QChan}(A, B) \rrbracket = p(\overline{M}(\llbracket A \rrbracket, F \llbracket B \rrbracket))) \rightarrow_{\overline{M}} ((\llbracket A \rrbracket \multimap_{\overline{M}} F \llbracket B \rrbracket) = (\llbracket A \rrbracket \multimap_{\overline{M}} F \llbracket B \rrbracket))$ defined in Subsection 4.5.1 and the unit $\eta(A) : A \rightarrow F(A)$ is the unit of the lifting monad F ; $\llbracket \text{QChan}(P, A) \rrbracket \multimap F \llbracket P \multimap A \rrbracket = \llbracket \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket$; and the second η also refers to the unit of the lifting monad F .

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\llbracket (\text{unbox}) \mid ! \Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket = \left(\left\{ \begin{array}{l} c \mapsto \emptyset \\ \mapsto \{f \mapsto (\emptyset, f)\} \\ \mapsto [\{f \mapsto [f_0]\}] \end{array} \right\}, \text{Diagram} \right) \quad (c)$$

where we let $\llbracket \text{QChan}(P, A) \rrbracket = p(\overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket))$, $f = (f_0, (f_p)_{p \in P}) \in \overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket)$, and

$$\begin{aligned}
\llbracket P \rrbracket &= (P, (P_p)_{p \in P}) \\
\llbracket A \rrbracket &= (X, (A_x)_{x \in X})
\end{aligned}$$

$$\bullet \frac{p \models P \quad \text{vBind}(!\Delta, \text{out}(Q), m, A)}{!\Delta \vdash (p, Q, m) : !\text{QChan}(P, A)} (\text{QChan}_I):$$

$$\begin{aligned} & \llbracket (\text{QChan}_I); (\tau_i)_{i:\text{leaf}} \mid !\Delta \vdash (p, Q, m) : !\text{QChan}(P, A) \rrbracket = \\ & \quad (\llbracket !\Delta \rrbracket = ![A_1] \otimes \cdots \otimes ![A_k]) \xrightarrow{\delta^{\otimes k}}_{\overline{M}} !! [A_1] \otimes \cdots \otimes !! [A_k] \\ & \quad \xrightarrow{\pi^{k-1}}_{\overline{M}} (![A_1] \otimes \cdots \otimes ![A_k]) = ![!\Delta] \xrightarrow{!(\eta_{\llbracket P \rrbracket} \llbracket !\Delta \rrbracket))}_{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} (\llbracket !\Delta \rrbracket \otimes \llbracket P \rrbracket)) \\ & \quad \xrightarrow{!(\llbracket P \rrbracket \multimap_{\overline{M}} \text{---})^h}_{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} F \llbracket A \rrbracket) \xrightarrow{\eta^{\circ \text{box}}}_{\overline{M}} F(!\llbracket \text{QChan}(P, A) \rrbracket) \end{aligned}$$

where δ refers to the comultiplication of $!$ -comonad and $\delta^{\otimes k} = \delta \llbracket A_1 \rrbracket \otimes \cdots \otimes \delta \llbracket A_k \rrbracket$; $\pi : !(-) \otimes !(-) \rightarrow_{\overline{M}} !(- \otimes -)$ is the strength of the monoidal comonad ($!$) and π^{k-1} represents application of π , $k-1$ times, as defined in Eq. 4.26; $\eta_{\llbracket P \rrbracket} \llbracket !\Delta \rrbracket$ refers to the natural transformation $\eta : \mathbf{1}_{\overline{M}} \rightarrow (\llbracket P \rrbracket \multimap -) \circ (- \otimes \llbracket P \rrbracket)$ and $(\llbracket P \rrbracket \multimap -)$ refers to the functor from Subsection 4.2.2; and $\eta \circ \text{box}$ is the composition of the morphism $\text{box} : (!\llbracket A \rrbracket \multimap_{\overline{M}} F \llbracket B \rrbracket) = (p \circ b)(\llbracket A \rrbracket \multimap_{\overline{M}} F \llbracket B \rrbracket) \rightarrow_{\overline{M}} (!\llbracket \text{QChan}(A, B) \rrbracket) = (p \circ b \circ p)(\overline{M}(\llbracket A \rrbracket, F \llbracket B \rrbracket))$ defined in Subsection 4.5.1 and the unit $\eta(A) : A \rightarrow F(A)$ is the unit of the lifting monad F .

Moreover, the morphism $h : \llbracket !\Delta \rrbracket \otimes \llbracket P \rrbracket \rightarrow_{\overline{M}} F \llbracket A \rrbracket$ is defined as follows:

$$h = \begin{aligned} & \llbracket !\Delta \rrbracket \otimes \llbracket P \rrbracket \xrightarrow{\text{id} \otimes q}_{\overline{M}} \llbracket !\Delta \rrbracket \otimes F \llbracket \text{out}(Q) \rrbracket \xrightarrow{t'}_{\overline{M}} F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \\ & \xrightarrow{F(f)}_{\overline{M}} F^2 \llbracket A \rrbracket \xrightarrow{\mu}_{\overline{M}} F \llbracket A \rrbracket \end{aligned}$$

where $t'_{A,B} = A \otimes FB \xrightarrow{\sigma}_{\overline{M}} FB \otimes A \xrightarrow{t_{B,A}} F(B \otimes A) \xrightarrow{F(\sigma)}_{\overline{M}} F(A \otimes B) : A \otimes FB \rightarrow_{\overline{M}} F(A \otimes B)$ is defined by using the tensorial strength $t_{B,A}$ of the branching computation monad F from Subsection 4.4.1; $\mu : F^2 \rightarrow F$ is the multiplication of branching monad from Subsection 4.4.1; and the morphisms q and f are defined as follows:

$$\begin{aligned} q &= \llbracket P \rrbracket \xrightarrow{\text{matching the wires in } P \text{ to } \text{in}(Q)}_{\overline{M}} \llbracket \text{in}(Q) \rrbracket \xrightarrow{\llbracket Q \rrbracket}_{\overline{M}} F \llbracket \text{out}(Q) \rrbracket \\ f &= \llbracket (\tau_i)_{i:\text{leaf}} \mid \text{vBind}(!\Delta, \text{out}(Q), m, A) \rrbracket : \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket \rightarrow_{\overline{M}} F \llbracket A \rrbracket \end{aligned}$$

To obtain the explicit form of the interpretation of this typing derivation, we first find the explicit representation of the morphism h . First, we let $\llbracket !\Delta \rrbracket = (C, (I)_C)$, $\llbracket P \rrbracket = (P, (P_p)_{p \in P})$, $\llbracket \text{in}(Q) \rrbracket = (\{\emptyset\}, (I_{\emptyset})_{\{\emptyset\}})$ where I_{\emptyset} does not refer to the unit I but the tensor product of $|\text{in}(Q)|$ -

qubits, $\llbracket \text{out}(Q) \rrbracket = (O, (O_o)_{o \in O})$, $\llbracket Q \rrbracket = (\{\emptyset \mapsto [o_1, \dots, o_l]\}, (Q_\emptyset))$, and

$$\begin{aligned}
 f &= \llbracket (\tau_i)_{i:\text{leaf}} \mid \text{vBind}(!\Delta, \text{out}(Q), m, A) \rrbracket \\
 &= (\{(c, o) \mapsto [x_1, \dots, x_{k(c,o)}]\}, (f_{c,o} : I \otimes O_o \rightarrow_{\overline{M}} \boxplus_{i=1..k(c,o)} A_{x_i}^{\otimes})_{(c,o)}) \\
 q &= (\{p \mapsto \emptyset \mapsto [o_1, \dots, o_l]\}, (\text{in}(Q))_{p \in P})
 \end{aligned}$$

Note that $\text{in}(Q)$ refers to the map from the variables in P to the variables in $\text{in}(Q)$ given by $p \models P$.

Then, we can represent the morphism h as follows.

$$h = \left(\left\{ \begin{aligned} (c, p) &\mapsto (c, [o_1, \dots, o_l]) \\ &\mapsto [(c, o_1), \dots, (c, o_l)] \\ &\mapsto [[x_1^1, \dots, x_{k^1}^1], \dots, [x_1^l, \dots, x_{k^l}^l]] \\ &\mapsto [x_1^1, \dots, x_{k^1}^1, \dots, x_1^l, \dots, x_{k^l}^l] \end{aligned} \right\}, \left(\begin{array}{c} \text{Diagram} \\ \text{Diagram} \\ \text{Diagram} \\ \text{Diagram} \end{array} \right)_{(c,p) \in C \times P} \right)$$

where we let $k^i = k^{(c,o_i)}$ for convenience.

Next, it follows from h that we can represent the morphism $(\llbracket P \rrbracket \multimap -)(h)$ as follows.

$$(\llbracket P \rrbracket \multimap -)(h) = \left(\left\{ \begin{aligned} f_p &= \{p' \mapsto (c, p)\} \\ &\mapsto \{p' \mapsto [x_1^1, \dots, x_{k^1}^1, \dots, x_1^l, \dots, x_{k^l}^l]\} \end{aligned} \right\}, \left(\begin{array}{c} \text{Diagram} \\ \text{Diagram} \\ \text{Diagram} \\ \text{Diagram} \end{array} \right)_{f_p \in P \rightarrow C \times P} \right)$$

Finally, we can represent the interpretation for each case for $!\Delta = \emptyset$ and $!\Delta \neq \emptyset$.

- $! \Delta = \emptyset$:

$$\begin{aligned}
& \llbracket (\text{QChan}_I); (\tau_i)_{i:\text{leaf}} \mid ! \Delta \vdash (p, Q, m) : ! \text{QChan}(P, A) \rrbracket \\
&= I \xrightarrow{\pi_I} \overline{M} ! I \xrightarrow{!(eta_{\llbracket P \rrbracket}(I))} \overline{M} !(\llbracket P \rrbracket \multimap (I \otimes \llbracket P \rrbracket)) \\
&\quad \xrightarrow{!(\llbracket P \rrbracket \multimap -)(h)} \overline{M} !(\llbracket P \rrbracket \multimap F \llbracket A \rrbracket)} \xrightarrow{\eta^{\text{box}}} F \llbracket \text{QChan}(P, A) \rrbracket \\
&= \left(\begin{array}{l}
\emptyset \mapsto (\emptyset, \text{id}_I) \\
\mapsto (\{p \mapsto (\emptyset, p)\}, \text{diagram}) \\
\mapsto (\{p \mapsto [x_1^1, \dots, x_{k^1}^1, \dots, x_1^l, \dots, x_{k^l}^l]\}, \text{diagram}) \\
\mapsto [(\{p \mapsto [x_1^1, \dots, x_{k^1}^1, \dots, x_1^l, \dots, x_{k^l}^l]\}, (\text{diagram})_{p \in P}), \text{id}_I]
\end{array} \right), (\text{id}_I)_{\{\emptyset\}}
\end{aligned}$$

where l is the number of branches in $\text{out}(Q)$ and k^i refers to the $k^{(\emptyset, o_i)}$ where $o_i \in O$ is from the denotation $\llbracket \text{out}(Q) \rrbracket = (O, (O_o)_{o \in O})$.

- $!\Delta \neq \emptyset$:

$$\begin{aligned}
& \llbracket (\text{QChan}_I); (\tau_i)_{i:\text{leaf}} \mid !\Delta \vdash (p, Q, m) : !\text{QChan}(P, A) \rrbracket \\
& = !\llbracket A_1 \rrbracket \otimes \cdots \otimes !\llbracket A_k \rrbracket \xrightarrow{\delta^\otimes} \overline{M} \llbracket !\llbracket A_1 \rrbracket \otimes \cdots \otimes !\llbracket A_k \rrbracket \rrbracket \xrightarrow{\pi^*} \overline{M} \llbracket !\Delta \rrbracket \\
& \xrightarrow{!(\eta \llbracket !\Delta \rrbracket)} \overline{M} \llbracket ([P] \multimap ([!\Delta] \otimes [P])) \rrbracket \xrightarrow{!([P] \multimap \text{--})(h)} \llbracket ([P] \multimap F[A]) \rrbracket \xrightarrow{\eta^{\text{obox}}} F \llbracket \text{QChan}(P, A) \rrbracket \\
& = \left(\begin{array}{l} c = ((x^1, m^1), \dots, (x^k, m^k)) \mapsto ((x^1, m^1), \text{id}_I), \dots, ((x^k, m^k), \text{id}_I) \\ \mapsto (((x^1, m^1), \dots, (x^k, m^k)), \text{id}_I) \\ \mapsto (\{p \mapsto (((x^1, m^1), \dots, (x^k, m^k)), p)\}, \text{diagram 1}) \\ \mapsto (\{p \mapsto [x_1^1, \dots, x_{k1}^1, \dots, x_1^l, \dots, x_{kl}^l]\}, \text{diagram 2}) \\ \mapsto [(\{p \mapsto [x_1^1, \dots, x_{k1}^1, \dots, x_1^l, \dots, x_{kl}^l]\}, (\text{diagram 3})_{p \in P}), \text{id}_I] \end{array} \right) , (\text{id}_I)_{c \in C}
\end{aligned}$$

where we let $!\Delta = !A_1, \dots, !A_k$, $!\llbracket \Delta \rrbracket = (C, (I)_C)$, $\llbracket A_i \rrbracket = (X^i, (A_{x^i}^i)_{x^i \in X^i})$, $(x^i, m^i) \in \sum_{x^i \in X^i} \overline{M}(I, A_{x^i}^i)$, $\llbracket P \rrbracket = (P, (P_p)_{p \in P})$, $\llbracket A \rrbracket = (X, (A_x)_{x \in X})$, l is the number of branches in $\text{out}(Q)$, and k^i refers to the $k^{(c, o_i)}$ where $o_i \in O$ is from the denotation $\llbracket \text{out}(Q) \rrbracket = (O, (O_o)_{o \in O})$.

$$\bullet \frac{\gamma_a \vdash m_a : A \quad \gamma_b \vdash m_b : A}{\gamma_a \times \gamma_b \vdash [m_a, m_b] : A} \text{(b):}$$

$$\llbracket \text{(b); } (\tau_a, \tau_b) \mid \gamma_a \times \gamma_b \vdash [m_a, m_b] : A \rrbracket = (\llbracket \gamma_a \times \gamma_b \rrbracket = \llbracket \gamma_a \rrbracket + \llbracket \gamma_b \rrbracket) \xrightarrow{(f_1, f_2)}_{\overline{M}} F \llbracket A \rrbracket$$

where $f_1 = \llbracket \tau_a \mid \gamma_a \vdash m_a : A \rrbracket : \llbracket \gamma_a \rrbracket \rightarrow_{\overline{M}} F \llbracket A \rrbracket$, $f_2 = \llbracket \tau_b \mid \gamma_b \vdash m_b : A \rrbracket : \llbracket \gamma_b \rrbracket \rightarrow_{\overline{M}} F \llbracket A \rrbracket$, and $(f_1, f_2) : \llbracket \gamma_a \rrbracket + \llbracket \gamma_b \rrbracket \rightarrow_{\overline{M}} F \llbracket A \rrbracket$ is defined from the coproduct of \overline{M} in Subsection 4.2.3.

Explicitly, the interpretation of the typing derivation is represented as follows:

$$\begin{aligned} & \llbracket \text{(b); } (\tau_a, \tau_b) \mid \gamma_a \times \gamma_b \vdash [m_a, m_b] : A \rrbracket \\ &= \left(\left\{ \begin{array}{l} (0, \gamma_a) \mapsto f_0^1(\gamma_a) \\ (1, \gamma_b) \mapsto f_0^2(\gamma_b) \end{array} \right\}, (f_{\gamma_a}^1)_{(0, \gamma_a)} \text{ :: } (f_{\gamma_b}^2)_{(1, \gamma_b)} \right) \end{aligned}$$

where we let $\llbracket \gamma_a \rrbracket = (\Gamma_a, (\Gamma_{\gamma_a}^a)_{\gamma_a \in \Gamma_a})$, $\llbracket \gamma_b \rrbracket = (\Gamma_b, (\Gamma_{\gamma_b}^b)_{\gamma_b \in \Gamma_b})$, $\llbracket A \rrbracket = (X, (A_x)_{x \in X})$ and

$$\begin{aligned} f_1 &= (f_0^1 : \Gamma_a \rightarrow \text{mset}(X), (f_{\gamma_a}^1 : \Gamma_{\gamma_a}^a \rightarrow \boxplus_{x \in f_0^1(\gamma_a)} A_x^{\otimes})_{\gamma_a \in \Gamma_a}) \\ f_2 &= (f_0^2 : \Gamma_b \rightarrow \text{mset}(X), (f_{\gamma_b}^2 : \Gamma_{\gamma_b}^b \rightarrow \boxplus_{x \in f_0^2(\gamma_b)} A_x^{\otimes})_{\gamma_b \in \Gamma_b}) \end{aligned}$$

4.5.4 . Interpretation of the extended typing relation

As each program has a side-effect, which buffers the quantum channel, to represent the program's precise meaning, we need to consider both the context and program. This was the reason why we defined the extended typing relation for circuit-buffering semantics in Definition 3.4.1. Now, we define the categorical interpretation of this extended typing relation.

Recall that a extended typing relation $! \Delta \vdash (Q, m) : A$ consists of a classical typing context $! \Delta$, a configuration (Q, m) , and a type A . A derivation of the extended typing relation $! \Delta \vdash (Q, m) : A$ implies that (Q, m) is valid and that $\forall i \text{ leaf} \cdot ! \Delta, \text{TC}_Q(\text{out}(Q)_i) \vdash m_i : A$, which implies that $! \Delta, \text{TC}_Q(\text{out}(Q)_i) \vdash m_i : A$ for each leaf i of the quantum channel Q , whose type derivations are denoted as τ_i . Therefore, from the denotations of each type judgement at the leaves, we can obtain a morphism $f_{vb} = \llbracket (\tau_i)_{i:\text{leaf}} \mid \text{vBind}(! \Delta, \text{out}(Q), m, A) \rrbracket : \llbracket ! \Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket \rightarrow_{\overline{M}_F} \llbracket A \rrbracket$ constructed by the vBind_{nb} and vBind_b rules.

Then, we define the interpretation of the extended typing relation as Eq (4.27).

$$\begin{aligned} & \llbracket (\tau_i)_{i:\text{leaf}} \mid ! \Delta \vdash (Q, m) : A \rrbracket \\ &= \llbracket ! \Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \xrightarrow{(\llbracket ! \Delta \rrbracket \otimes -)(\llbracket Q \rrbracket)}_{\overline{M}} \llbracket ! \Delta \rrbracket \otimes F \llbracket \text{out}(Q) \rrbracket \\ & \xrightarrow{t'}_{\overline{M}} F(\llbracket ! \Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \xrightarrow{F(f_{vb})}_{\overline{M}} F^2 \llbracket A \rrbracket \xrightarrow{\mu}_{\overline{M}} F \llbracket A \rrbracket \end{aligned} \quad (4.27)$$

where $\llbracket Q \rrbracket : \llbracket \text{in}(Q) \rrbracket \rightarrow_{\overline{M}_F} \llbracket \text{out}(Q) \rrbracket$ is the interpretation of the quantum channel Q and $f_{vb} = \llbracket (\tau_i)_{i:\text{leaf}} \mid \text{vBind}(! \Delta, \text{out}(Q), m, A) \rrbracket$.

4.5.5 . Examples–non-trivial quantum channel and teleportation

Now, we show how the interpretation works on a typing derivation for the programs exp in Example 3.1.1 and tel in Example 3.1.2. In each case, we first take the typing derivation which are shown in Example 3.2.3 and Example 3.2.4, then present the interpretation of the typing derivation.

Example 4.5.2. For the program exp , let's consider the typing derivation that we have shown in Example 3.2.3. It is represented in Figure 4.29. where we let

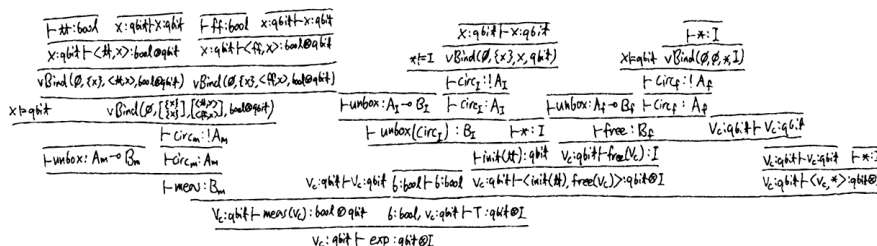


Figure 4.29: Type derivation τ_{exp} from Example 3.2.3

$$\begin{aligned}
 \text{exp} &= \mathbf{let} \langle b, v_c \rangle = \mathbf{meas}(v_c) \mathbf{in} \quad T \\
 T &= \mathbf{if} \, b \mathbf{then} \langle \mathbf{init}(tt), \mathbf{free}(v_c) \rangle \mathbf{else} \langle v_c, * \rangle \\
 A_m &= \mathbf{QChan}(\mathbf{qubit}, \mathbf{bool} \otimes \mathbf{qubit}) \\
 B_m &= \mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{qubit} \\
 \text{circ}_m &= \left(x, x \text{ --- } \boxed{\text{meas}} \begin{matrix} \curvearrowleft x \\ \curvearrowright x \end{matrix}, \bullet \begin{matrix} \curvearrowright \langle tt, x \rangle \\ \curvearrowleft \langle ff, x \rangle \end{matrix} \right) \\
 A_f &= \mathbf{QChan}(\mathbf{qubit}, I) \\
 B_f &= \mathbf{qubit} \multimap I \\
 \text{circ}_f &= \left(x, x \text{ --- } \boxed{\text{free } x} \bullet, \bullet \text{ --- } * \right) \\
 \mathbf{free}(v_c) &= \mathbf{unbox}(\text{circ}_f)(v_c) \\
 A_I &= \mathbf{QChan}(I, \mathbf{qubit}) \\
 B_I &= I \multimap \mathbf{qubit} \\
 \text{circ}_I &= \left(*, * \text{ --- } \boxed{\text{init } b \, x} \bullet \text{ --- } x \right) \\
 \mathbf{init}(tt) &= \mathbf{unbox}(\text{circ}_I)(*)
 \end{aligned}$$

Then, from the definition of extended typing judgement, we can obtain the following derivation:

$$\frac{(\epsilon(v_c), \text{exp}) \text{ valid} \quad \frac{\vdots}{v_c : \mathbf{qubit} \vdash \text{exp} : \mathbf{qubit} \otimes I} (\tau_{\text{exp}})}{\vdash (\epsilon(v_c), \text{exp}) : \mathbf{qubit} \otimes I}$$

According to the type derivation and interpretation of the extended typing

judgement and the typing derivation rules, we are going to show that

$$\llbracket \tau_{exp} \vdash (\epsilon(v_c), exp) : \mathbf{qubit} \otimes I \rrbracket =$$

$$\left(\left\{ \begin{array}{l} \emptyset \mapsto [\emptyset] \\ \mapsto [(\emptyset, \emptyset)] \\ \mapsto [[(\emptyset), (\emptyset)]] \\ \mapsto [(\emptyset, \emptyset), (\emptyset, \emptyset)] \end{array} \right\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \\ \text{Diagram 3} \end{array} \right) \right)$$

For simplicity, we ignore, in this example, the typing derivation for the interpretation of each type judgement which is supposed to be the part of the typing derivation τ_{exp} .

1. Interpretations of some preliminary typing derivations:

$$\llbracket \vdash * : I \rrbracket = (\{\emptyset \mapsto [\emptyset]\}, (id_I)_{\{\emptyset\}}) \quad (4.28)$$

$$\llbracket \vdash tt : bool \rrbracket = (\{\emptyset \mapsto [tt]\}, (id_I)_{\{\emptyset\}}) \quad (4.29)$$

$$\llbracket \vdash ff : bool \rrbracket = (\{\emptyset \mapsto [ff]\}, (id_I)_{\{\emptyset\}}) \quad (4.30)$$

$$\llbracket x : \mathbf{qubit} \vdash x : \mathbf{qubit} \rrbracket = (\{\emptyset \mapsto [\emptyset]\}, (id_{[q]})_{\{\emptyset\}}) \quad (4.31)$$

$$\llbracket v_c : \mathbf{qubit} \vdash v_c : \mathbf{qubit} \rrbracket = (\{\emptyset \mapsto [\emptyset]\}, (id_{[q]})_{\{\emptyset\}}) \quad (4.32)$$

$$\llbracket b : bool \vdash b : bool \rrbracket = (\{\langle \emptyset, x \rangle \mapsto [x] \mid x \in \{tt, ff\}\}, (id_I)_{\{\emptyset\} \times \{tt, ff\}}) \quad (4.33)$$

$$\llbracket v_c : \mathbf{qubit} \vdash \langle v_c, * \rangle : \mathbf{qubit} \otimes I \rrbracket = \left(\{\emptyset \mapsto [(\emptyset, \emptyset)]\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \right)_{\{\emptyset\}} \right) \quad (4.34)$$

$$\llbracket x : \mathbf{qubit} \vdash \langle tt, x \rangle : bool \otimes \mathbf{qubit} \rrbracket = \left(\{\langle \emptyset, \emptyset \rangle \mapsto [(tt, \emptyset)]\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \right)_{\{\langle \emptyset, \emptyset \rangle\}} \right) \quad (4.35)$$

$$\llbracket x : \mathbf{qubit} \vdash \langle ff, x \rangle : bool \otimes \mathbf{qubit} \rrbracket = \left(\{\langle \emptyset, \emptyset \rangle \mapsto [(ff, \emptyset)]\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \right)_{\{\langle \emptyset, \emptyset \rangle\}} \right)$$

$$\llbracket vBind(\emptyset, \emptyset, *, I) \rrbracket = \llbracket \vdash * : I \rrbracket \quad (4.36)$$

$$\llbracket vBind(\emptyset, \{x\}, x, \mathbf{qubit}) \rrbracket = \llbracket x : \mathbf{qubit} \vdash x : \mathbf{qubit} \rrbracket \quad (4.37)$$

$$\llbracket vBind(\emptyset, \{x\}, \langle tt, x \rangle, bool \otimes \mathbf{qubit}) \rrbracket = \llbracket x : \mathbf{qubit} \vdash \langle tt, x \rangle : bool \otimes \mathbf{qubit} \rrbracket \quad (4.38)$$

$$\llbracket vBind(\emptyset, \{x\}, \langle ff, x \rangle, bool \otimes \mathbf{qubit}) \rrbracket = \llbracket x : \mathbf{qubit} \vdash \langle ff, x \rangle : bool \otimes \mathbf{qubit} \rrbracket$$

$$\llbracket vBind(\emptyset, [\{x\}], [\{x\}], [\langle tt, x \rangle, \langle ff, x \rangle], bool \otimes \mathbf{qubit}) \rrbracket =$$

$$\left(\left\{ \begin{array}{l} \langle \emptyset, (0, \emptyset) \rangle \mapsto (0, (\emptyset, \emptyset)) \mapsto [(ff, \emptyset)] \\ \langle \emptyset, (1, \emptyset) \rangle \mapsto (1, (\emptyset, \emptyset)) \mapsto [(ff, \emptyset)] \end{array} \right\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \right)_{\{\langle \emptyset, (0, \emptyset) \rangle\}} \quad \vdots \quad \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \right)_{\{\langle \emptyset, (1, \emptyset) \rangle\}} \right) \quad (4.39)$$

3. Interpretation of the typing derivation of $\vdash \text{init}(tt) : \mathbf{qubit}$:

$$\llbracket \text{init } tt \ x \ \epsilon(\{x\}) \ \epsilon(\{x\}) \rrbracket = \left(\{\emptyset \mapsto [(0, \emptyset), (1, \emptyset)], \left(\overset{\uparrow \mathbf{q}}{\text{init } \#} \right)_{\{\emptyset\}} \right) \quad (4.46)$$

$$\begin{aligned} \llbracket \vdash \text{circ}_I : !A_I \rrbracket = \\ \left(\left\{ \emptyset \mapsto \left[\left(\left(\{\emptyset \mapsto [\emptyset]\}, \left(\overset{\uparrow \mathbf{q}}{\text{init } \#} \right)_{\{\emptyset\}} \right) \right], id_I \right] \right\}, (id_I)_{\{\emptyset\}} \right) \end{aligned} \quad (4.47)$$

$$\llbracket \vdash \text{circ}_I : A_I \rrbracket = \left(\left\{ \emptyset \mapsto \left[\left(\{\emptyset \mapsto [\emptyset]\}, \left(\overset{\uparrow \mathbf{q}}{\text{init } \#} \right)_{\{\emptyset\}} \right) \right] \right\}, (id_I)_{\{\emptyset\}} \right) \quad (4.48)$$

$$\begin{aligned} \llbracket \vdash \text{unbox} : A_I \multimap B_I \rrbracket = \\ \left(\left\{ \emptyset \mapsto \left[\left\{ \begin{array}{l} (f_0, (f_\emptyset)) \mapsto [f_0] \\ |(f_0, (f_\emptyset)) \in \overline{M}_F(I, \llbracket \mathbf{qubit} \rrbracket)| \end{array} \right\} \right] \right\}, \left(\text{Diagram} \right)_{\{\emptyset\}} \right) \end{aligned} \quad (4.49)$$

$$\llbracket \vdash \text{unbox}(\text{circ}_I) : B_I \rrbracket = \left(\left\{ \emptyset \mapsto [\{\emptyset \mapsto [\emptyset]\}] \right\}, \left(\text{Diagram} \right)_{\{\emptyset\}} \right) \quad (4.50)$$

$$\llbracket \vdash \text{init}(tt) : \mathbf{qubit} \rrbracket = \left(\left\{ \emptyset \mapsto [\emptyset] \right\}, \left(\text{Diagram} \right)_{\{\emptyset\}} \right) \quad (4.51)$$

4. Interpretation of the typing derivation of $\vdash \text{free}(v_c) : I$

$$\llbracket \text{free } x \in (\emptyset) \rrbracket = \left(\{\emptyset \mapsto [\emptyset]\}, \left(\frac{\text{free}}{\top} \right)_{\{\emptyset\}} \right) \quad (4.52)$$

$$\begin{aligned} & \llbracket \vdash \text{circ}_f : !A_f \rrbracket = \\ & \left(\left\{ \emptyset \mapsto \left[\left(\left(\{\emptyset \mapsto [\emptyset]\}, \left(\frac{\text{free}}{\top} \right)_{\{\emptyset\}} \right) \right), id_I \right] \right\}, (id_I)_{\{\emptyset\}} \right) \end{aligned} \quad (4.53)$$

$$\llbracket \vdash \text{circ}_f : A_f \rrbracket = \left(\left\{ \emptyset \mapsto \left[\left(\{\emptyset \mapsto [\emptyset]\}, \left(\frac{\text{free}}{\top} \right)_{\{\emptyset\}} \right) \right] \right\}, (id_I)_{\{\emptyset\}} \right) \quad (4.54)$$

$$\begin{aligned} & \llbracket \vdash \text{unbox} : A_f \multimap B_f \rrbracket = \\ & \left(\left\{ \emptyset \mapsto \left[\left\{ (f_0, (f_\emptyset)) \mapsto [f_0] \right. \right. \right. \right. \\ & \quad \left. \left. \left. \left. \mid (f_0, (f_\emptyset)) \in \overline{M}_F(\llbracket \mathbf{qubit} \rrbracket, I) \right\} \right\} \right\}, \left(\frac{\uparrow \text{free} [I \multimap (q \multimap \text{circ}_f I)]}{\text{unbox}} \right)_{\{\emptyset\}} \right) \end{aligned} \quad (4.55)$$

$$\llbracket \vdash \text{free} : B_f \rrbracket = \left(\left\{ \emptyset \mapsto [\{\emptyset \mapsto [\emptyset]\}] \right\}, \left(\frac{\uparrow \text{free} [I \multimap (q \multimap \text{circ}_f I)]}{\text{free}} \right)_{\{\emptyset\}} \right) \quad (4.56)$$

$$\llbracket \vdash \text{free}(v_c) : I \rrbracket = \left(\left\{ \emptyset \mapsto [\emptyset] \right\}, \left(\frac{\uparrow \text{free}}{I} \right)_{\{\emptyset\}} \right) \quad (4.57)$$

5. Interpretation of the typing derivation of $b : \text{bool}, v_c : \mathbf{qubit} \vdash T : \mathbf{qubit} \otimes I$

We are going to interpret the typing derivation in Figure 4.30 using the interpretation of the type derivation π_4 of $\text{init}(tt)$ and the type derivation π_5 $\text{free}(v_c)$ which was shown before.

$$\frac{\frac{\frac{\pi_4}{\vdash \text{init}(tt) : \mathbf{qbit}} \quad \frac{\pi_5}{v_c : \mathbf{qbit} \vdash \text{free}(v_c) : I} \quad \frac{v_c : \mathbf{qbit} \vdash v_c : \mathbf{qbit} \quad \vdash * : I}{v_c : \mathbf{qbit} \vdash \langle v_c, * \rangle : \mathbf{qbit} \otimes I}}{v_c : \mathbf{qbit} \vdash \langle \text{init}(tt), \text{free}(v_c) \rangle : \mathbf{qbit} \otimes I}}{b : \text{bool} \vdash b : \text{bool} \quad v_c : \mathbf{qbit} \vdash \langle \text{init}(tt), \text{free}(v_c) \rangle : \mathbf{qbit} \otimes I}}{b : \text{bool}, v_c : \mathbf{qbit} \vdash T : \mathbf{qbit} \otimes I}$$

Figure 4.30: Typing derivation of the part T

$$\llbracket v_c : \mathbf{qubit} \vdash \langle \text{init}(tt), \text{free}(v_c) \rangle : \mathbf{qubit} \otimes I \rrbracket = \left(\left\{ \emptyset \mapsto [(\emptyset, \emptyset)] \right\}, \left(\begin{array}{c} \uparrow \mathbf{qbit} \\ \text{init } \# \quad \text{free} \\ \uparrow \quad \uparrow \\ \text{I} \end{array} \right)_{\{\emptyset\}} \right) \quad (4.58)$$

$$\begin{aligned} \llbracket b : \text{bool}, v_c : \mathbf{qubit} \vdash T : \mathbf{qubit} \otimes I \rrbracket = \\ \left(\left\{ \begin{array}{l} (\emptyset, tt, \emptyset) \mapsto [(\emptyset, \emptyset)] \\ (\emptyset, ff, \emptyset) \mapsto [(\emptyset, \emptyset)] \end{array} \right\}, \left(\begin{array}{c} \uparrow \mathbf{qbit} \\ \text{init } \# \quad \text{free} \\ \uparrow \quad \uparrow \\ \text{I} \end{array} \right)_{\{(\emptyset, tt, \emptyset)\}} \quad :: \quad \left(\begin{array}{c} \uparrow \mathbf{qbit} \\ \text{free} \\ \uparrow \\ \text{I} \end{array} \right)_{\{(\emptyset, ff, \emptyset)\}} \right) \quad (4.59) \end{aligned}$$

6. Interpretation of the typing derivation of $v_c : \mathbf{qubit} \vdash \text{exp} : \mathbf{qubit} \otimes I$

From the interpretation of T in Eq. (4.59) and $\text{meas}(v_c)$ in Eq. (4.45), we can obtain the interpretation of exp .

$$\llbracket v_c : \mathbf{qubit} \vdash \text{exp} : \mathbf{qubit} \otimes I \rrbracket = \left(\left\{ \emptyset \mapsto [(\emptyset, \emptyset), (\emptyset, \emptyset)] \right\}, \left(\begin{array}{c} \uparrow \mathbf{qbit} \\ \text{init } \# \quad \text{free} \\ \uparrow \quad \uparrow \\ \text{I} \end{array} \right)_{\{\emptyset\}} \right) \quad (4.60)$$

7. Interpretation of the extended typing derivation of $\vdash (\epsilon(v_c), \text{exp}) : \mathbf{qubit} \otimes I$

$$\llbracket \epsilon(v_c) \rrbracket = (\{\emptyset \mapsto [\emptyset]\}, (\text{id}_{[q]})_{\{\emptyset\}}) \quad (4.61)$$

$$\llbracket v\text{Bind}(\emptyset, \{v_c\}, \text{exp}, \mathbf{qubit} \otimes I) \rrbracket = \llbracket v_c : \mathbf{qubit} \vdash \text{exp} : \mathbf{qubit} \otimes I \rrbracket \quad (4.62)$$

$$\llbracket \vdash (\epsilon(v_c), \text{exp}) : \mathbf{qubit} \otimes I \rrbracket = \left(\left\{ \begin{array}{l} \emptyset \mapsto [\emptyset \mapsto [(\emptyset, \emptyset)] \\ \mapsto [[(\emptyset, \emptyset), (\emptyset, \emptyset)]] \\ \mapsto [(\emptyset, \emptyset), (\emptyset, \emptyset)] \end{array} \right\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \end{array} \right) \right)_{\{\emptyset\}} \quad (4.63)$$

Example 4.5.3. Next, for the teleportation, we rely on the typing derivation that has been shown in Example 3.2.4. Let's denote the typing derivation as τ_{tel} . We are going to show that the interpretation of this typing derivation results in Eq. (4.93). Note that we omit in this example the typing derivations in the interpretation symbols which correspond to the sub-trees of the typing derivation τ_{tel} .

$$\llbracket \tau_{tel} \vdash tel : (\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool}) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit}) \rrbracket = \left(\left\{ \emptyset \mapsto \left[\left(\left\{ \emptyset \mapsto \left[\begin{array}{l} (tt, tt), (tt, ff), \\ (ff, tt), (ff, ff) \end{array} \right] \right\}, \left\{ \begin{array}{l} (b_x, b_y) \mapsto [\emptyset] \\ | b_x, b_y \in \{tt, ff\} \end{array} \right\} \right] \right\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \\ \text{Diagram 3} \\ \text{Diagram 4} \\ \text{Diagram 5} \\ \text{Diagram 6} \\ \text{Diagram 7} \\ \text{Diagram 8} \\ \text{Diagram 9} \\ \text{Diagram 10} \\ \text{Diagram 11} \\ \text{Diagram 12} \\ \text{Diagram 13} \\ \text{Diagram 14} \\ \text{Diagram 15} \\ \text{Diagram 16} \\ \text{Diagram 17} \\ \text{Diagram 18} \\ \text{Diagram 19} \\ \text{Diagram 20} \\ \text{Diagram 21} \\ \text{Diagram 22} \\ \text{Diagram 23} \\ \text{Diagram 24} \\ \text{Diagram 25} \\ \text{Diagram 26} \\ \text{Diagram 27} \\ \text{Diagram 28} \\ \text{Diagram 29} \\ \text{Diagram 30} \\ \text{Diagram 31} \\ \text{Diagram 32} \\ \text{Diagram 33} \\ \text{Diagram 34} \\ \text{Diagram 35} \\ \text{Diagram 36} \\ \text{Diagram 37} \\ \text{Diagram 38} \\ \text{Diagram 39} \\ \text{Diagram 40} \\ \text{Diagram 41} \\ \text{Diagram 42} \\ \text{Diagram 43} \\ \text{Diagram 44} \\ \text{Diagram 45} \\ \text{Diagram 46} \\ \text{Diagram 47} \\ \text{Diagram 48} \\ \text{Diagram 49} \\ \text{Diagram 50} \\ \text{Diagram 51} \\ \text{Diagram 52} \\ \text{Diagram 53} \\ \text{Diagram 54} \\ \text{Diagram 55} \\ \text{Diagram 56} \\ \text{Diagram 57} \\ \text{Diagram 58} \\ \text{Diagram 59} \\ \text{Diagram 60} \\ \text{Diagram 61} \\ \text{Diagram 62} \\ \text{Diagram 63} \\ \text{Diagram 64} \\ \text{Diagram 65} \\ \text{Diagram 66} \\ \text{Diagram 67} \\ \text{Diagram 68} \\ \text{Diagram 69} \\ \text{Diagram 70} \\ \text{Diagram 71} \\ \text{Diagram 72} \\ \text{Diagram 73} \\ \text{Diagram 74} \\ \text{Diagram 75} \\ \text{Diagram 76} \\ \text{Diagram 77} \\ \text{Diagram 78} \\ \text{Diagram 79} \\ \text{Diagram 80} \\ \text{Diagram 81} \\ \text{Diagram 82} \\ \text{Diagram 83} \\ \text{Diagram 84} \\ \text{Diagram 85} \\ \text{Diagram 86} \\ \text{Diagram 87} \\ \text{Diagram 88} \\ \text{Diagram 89} \\ \text{Diagram 90} \\ \text{Diagram 91} \\ \text{Diagram 92} \\ \text{Diagram 93} \\ \text{Diagram 94} \\ \text{Diagram 95} \\ \text{Diagram 96} \\ \text{Diagram 97} \\ \text{Diagram 98} \\ \text{Diagram 99} \\ \text{Diagram 100} \end{array} \right) \right)_{\{\emptyset\}} \quad (4.64)$$

Now, we show how to obtain the interpretation as follows.

1. Interpretation of the typing derivation of elementary quantum channel constants

$$\llbracket \vdash H : \mathbf{qubit} \multimap \mathbf{qubit} \rrbracket = \left(\left\{ \emptyset \mapsto [\{\emptyset \mapsto [\emptyset]\}] \right\}, \left(\begin{array}{c} \text{Diagram 1} \\ \text{Diagram 2} \\ \text{Diagram 3} \\ \text{Diagram 4} \\ \text{Diagram 5} \\ \text{Diagram 6} \\ \text{Diagram 7} \\ \text{Diagram 8} \\ \text{Diagram 9} \\ \text{Diagram 10} \\ \text{Diagram 11} \\ \text{Diagram 12} \\ \text{Diagram 13} \\ \text{Diagram 14} \\ \text{Diagram 15} \\ \text{Diagram 16} \\ \text{Diagram 17} \\ \text{Diagram 18} \\ \text{Diagram 19} \\ \text{Diagram 20} \\ \text{Diagram 21} \\ \text{Diagram 22} \\ \text{Diagram 23} \\ \text{Diagram 24} \\ \text{Diagram 25} \\ \text{Diagram 26} \\ \text{Diagram 27} \\ \text{Diagram 28} \\ \text{Diagram 29} \\ \text{Diagram 30} \\ \text{Diagram 31} \\ \text{Diagram 32} \\ \text{Diagram 33} \\ \text{Diagram 34} \\ \text{Diagram 35} \\ \text{Diagram 36} \\ \text{Diagram 37} \\ \text{Diagram 38} \\ \text{Diagram 39} \\ \text{Diagram 40} \\ \text{Diagram 41} \\ \text{Diagram 42} \\ \text{Diagram 43} \\ \text{Diagram 44} \\ \text{Diagram 45} \\ \text{Diagram 46} \\ \text{Diagram 47} \\ \text{Diagram 48} \\ \text{Diagram 49} \\ \text{Diagram 50} \\ \text{Diagram 51} \\ \text{Diagram 52} \\ \text{Diagram 53} \\ \text{Diagram 54} \\ \text{Diagram 55} \\ \text{Diagram 56} \\ \text{Diagram 57} \\ \text{Diagram 58} \\ \text{Diagram 59} \\ \text{Diagram 60} \\ \text{Diagram 61} \\ \text{Diagram 62} \\ \text{Diagram 63} \\ \text{Diagram 64} \\ \text{Diagram 65} \\ \text{Diagram 66} \\ \text{Diagram 67} \\ \text{Diagram 68} \\ \text{Diagram 69} \\ \text{Diagram 70} \\ \text{Diagram 71} \\ \text{Diagram 72} \\ \text{Diagram 73} \\ \text{Diagram 74} \\ \text{Diagram 75} \\ \text{Diagram 76} \\ \text{Diagram 77} \\ \text{Diagram 78} \\ \text{Diagram 79} \\ \text{Diagram 80} \\ \text{Diagram 81} \\ \text{Diagram 82} \\ \text{Diagram 83} \\ \text{Diagram 84} \\ \text{Diagram 85} \\ \text{Diagram 86} \\ \text{Diagram 87} \\ \text{Diagram 88} \\ \text{Diagram 89} \\ \text{Diagram 90} \\ \text{Diagram 91} \\ \text{Diagram 92} \\ \text{Diagram 93} \\ \text{Diagram 94} \\ \text{Diagram 95} \\ \text{Diagram 96} \\ \text{Diagram 97} \\ \text{Diagram 98} \\ \text{Diagram 99} \\ \text{Diagram 100} \end{array} \right) \right)_{\{\emptyset\}} \quad (4.65)$$

$$\llbracket x : \mathbf{qubit} \vdash H(x) : \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\emptyset]\}, \left(\begin{array}{c} \uparrow \\ \boxed{H} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.66)$$

$$\llbracket \vdash H(\mathit{init}(tt)) : \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\emptyset]\}, \left(\begin{array}{c} \uparrow \\ \boxed{H} \\ \downarrow \\ \mathit{init} \# \end{array} \right)_{\{\emptyset\}} \right) \quad (4.67)$$

$$\llbracket \vdash \langle H(\mathit{init}(tt)), \mathit{init}(tt) \rangle : \mathbf{qubit} \otimes \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [(\emptyset, \emptyset)]\}, \left(\begin{array}{c} \uparrow \downarrow \\ \boxed{H} \quad \boxed{\mathit{init}} \\ \downarrow \quad \downarrow \\ \mathit{init} \# \quad \mathit{init} \# \end{array} \right)_{\{\emptyset\}} \right) \quad (4.68)$$

$$\llbracket \vdash X : \mathbf{qubit} \multimap \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\{\emptyset \mapsto [\emptyset]\}]\}, \left(\begin{array}{c} \uparrow \\ \boxed{X} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.69)$$

$$\llbracket q : \mathbf{qubit} \vdash X(q) : \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\emptyset]\}, \left(\begin{array}{c} \uparrow \\ \boxed{X} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.70)$$

$$\llbracket \vdash Z : \mathbf{qubit} \multimap \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\{\emptyset \mapsto [\emptyset]\}]\}, \left(\begin{array}{c} \uparrow \\ \boxed{Z} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.71)$$

$$\llbracket q : \mathbf{qubit} \vdash Z(q) : \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\emptyset]\}, \left(\begin{array}{c} \uparrow \\ \boxed{Z} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.72)$$

$$\llbracket q : \mathbf{qubit} \vdash X(Z(q)) : \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\emptyset]\}, \left(\begin{array}{c} \uparrow \\ \boxed{X} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.73)$$

$$\llbracket \vdash \mathit{CNOT} : \mathbf{qubit} \otimes \mathbf{qubit} \multimap \mathbf{qubit} \otimes \mathbf{qubit} \rrbracket = \left(\{\emptyset \mapsto [\{(\emptyset, \emptyset) \mapsto [(\emptyset, \emptyset)]\}]\}, \left(\begin{array}{c} \uparrow \downarrow \\ \boxed{\mathit{CNOT}} \\ \downarrow \quad \downarrow \\ \emptyset \quad \emptyset \end{array} \right)_{\{(\emptyset, \emptyset)\}} \right) \quad (4.74)$$

$$\llbracket y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \mathit{CNOT}(x, y) : \mathbf{qubit} \otimes \mathbf{qubit} \rrbracket = \left(\{(\emptyset, \emptyset) \mapsto [(\emptyset, \emptyset)]\}, \left(\begin{array}{c} \uparrow \\ \boxed{\mathit{CNOT}} \\ \downarrow \\ \emptyset \end{array} \right)_{\{\emptyset\}} \right) \quad (4.75)$$

$$\llbracket \vdash \text{meas}_f : B_m \rrbracket = \left(\left\{ \emptyset \mapsto [\{\emptyset \mapsto [tt, ff]\}] \right\}, \left(\begin{array}{c} \text{Diagram} \\ \{ \emptyset \} \end{array} \right) \right) \quad (4.76)$$

$$\llbracket y : \mathbf{qubit} \vdash \text{meas}_f(y) : \text{bool} \rrbracket = \left(\left\{ \emptyset \mapsto [tt, ff] \right\}, \left(\begin{array}{c} \text{Diagram} \\ \{ \emptyset \} \end{array} \right) \right) \quad (4.77)$$

$$\llbracket x : \mathbf{qubit} \vdash \text{meas}_f(H(x)) : \text{bool} \rrbracket = \left(\left\{ \emptyset \mapsto [tt, ff] \right\}, \left(\begin{array}{c} \text{Diagram} \\ \{ \emptyset \} \end{array} \right) \right) \quad (4.78)$$

2. Interpretation of the typing derivation of $\vdash A : \mathbf{qubit} \multimap (\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool})$ where $A = \lambda y. \lambda x. (\mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle)$

$$\llbracket x : \mathbf{qubit}, y : \mathbf{qubit} \vdash \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle : \mathbf{bool} \otimes \mathbf{bool} \rrbracket = \left(\{(\emptyset, \emptyset) \mapsto [(tt, tt), (tt, ff), (ff, tt), (ff, ff)]\}, \left(\begin{array}{c} \text{Circuit diagram for } \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle \\ \text{with } \{(\emptyset, \emptyset)\} \end{array} \right) \right) \quad (4.79)$$

$$\left[\frac{y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \mathbf{CNOT} \langle x, y \rangle : \mathbf{QQ} \quad y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle : \mathbf{BB}}{y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle : \mathbf{bool} \otimes \mathbf{bool}} \right] = \left(\{(\emptyset, \emptyset) \mapsto [(tt, tt), (tt, ff), (ff, tt), (ff, ff)]\}, \left(\begin{array}{c} \text{Circuit diagram for } \mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle \\ \text{with } \{(\emptyset, \emptyset)\} \end{array} \right) \right) \quad (4.80)$$

$$\left[\frac{y : \mathbf{qubit}, x : \mathbf{qubit} \vdash \mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle : \mathbf{bool} \otimes \mathbf{bool}}{y : \mathbf{qubit} \vdash \lambda x. (\mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle) : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})} \right] = \left(\{\emptyset \mapsto [\{\emptyset \mapsto [(tt, tt), (tt, ff), (ff, tt), (ff, ff)]\}]\}, \left(\begin{array}{c} \text{Circuit diagram for } \lambda x. (\mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle) \\ \text{with } \{\emptyset\} \end{array} \right) \right) \quad (4.81)$$

$$\left[\frac{y : \mathbf{qubit} \vdash \lambda x. (\mathbf{let} \langle x, y \rangle = \mathbf{CNOT} \langle x, y \rangle \mathbf{in} \langle \mathbf{meas}(H(x)), \mathbf{meas}(y) \rangle) : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})}{\vdash A : \mathbf{qubit} \multimap (\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool})} \right] =$$

$$\left(\left\{ \emptyset \mapsto [\{\emptyset \mapsto [\{\emptyset \mapsto [(tt, tt), (tt, ff), (ff, tt), (ff, ff)]\}]] \right\}, \right.$$
$$\left. \right)_{\{\emptyset\}}$$
(4.82)

3. Interpretation of the typing derivation of $\vdash B : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})$ where $B = \lambda q. \lambda xy. (\mathbf{let} \langle x, y \rangle = xy \mathbf{in} U_{XY})$

$$\left[\frac{y : \mathbf{bool} \vdash y : \mathbf{bool} \quad q : \mathbf{qubit} \vdash X(q) : \mathbf{qubit} \quad q : \mathbf{qubit} \vdash q : \mathbf{qubit}}{y : \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{if} \ y \ \mathbf{then} \ X(q) \ \mathbf{else} \ q : \mathbf{qubit}} \right] =$$

$$\left(\left\{ (b, \emptyset) \mapsto [\emptyset] \right\}, \left(\begin{array}{c} \uparrow \\ \otimes \\ \uparrow \\ \hline \end{array} \right)_{\{(tt, \emptyset)\}} \quad \vdash \quad \left(\begin{array}{c} \uparrow \\ q \\ \hline \end{array} \right)_{\{(ff, \emptyset)\}} \right)$$
(4.83)

$$\left[\frac{y : \mathbf{bool} \vdash y : \mathbf{bool} \quad q : \mathbf{qubit} \vdash X(Z(q)) : \mathbf{qubit} \quad q : \mathbf{qubit} \vdash Z(q) : \mathbf{qubit}}{y : \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{if} \ y \ \mathbf{then} \ X(Z(q)) \ \mathbf{else} \ Z(q) : \mathbf{qubit}} \right] =$$

$$\left(\left\{ (b, \emptyset) \mapsto [\emptyset] \right\}, \left(\begin{array}{c} \uparrow \\ \otimes \\ \uparrow \\ \hline \end{array} \right)_{\{(tt, \emptyset)\}} \quad \vdash \quad \left(\begin{array}{c} \uparrow \\ Z \\ \hline \end{array} \right)_{\{(ff, \emptyset)\}} \right)$$
(4.84)

$$\left[\frac{x : \mathbf{bool} \vdash x : \mathbf{bool} \quad y : \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{CTRL}_1 : \mathbf{qubit} \quad y : \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{CTRL}_2 : \mathbf{qubit}}{x : \mathbf{qubit}, y : \mathbf{qubit}, q : \mathbf{qubit} \vdash U_{XY} : \mathbf{qubit}} \right] =$$

$$\left(\left\{ (\emptyset, b_x, b_y) \mapsto [\emptyset] \right\}, \left(\begin{array}{c} \uparrow \\ \otimes \\ \uparrow \\ \hline \end{array} \right)_{\{(\emptyset, tt, tt)\}} \quad \vdash \quad \left(\begin{array}{c} \uparrow \\ Z \\ \hline \end{array} \right)_{\{(\emptyset, tt, ff)\}} \quad \vdash \quad \left(\begin{array}{c} \uparrow \\ \otimes \\ \uparrow \\ \hline \end{array} \right)_{\{(\emptyset, ff, tt)\}} \quad \vdash \quad \left(\begin{array}{c} \uparrow \\ q \\ \hline \end{array} \right)_{\{(\emptyset, ff, ff)\}} \right)$$
(4.85)

where $U_{XY} = \mathbf{if} \ x \ \mathbf{then} \ (\mathbf{if} \ y \ \mathbf{then} \ X(Z(q)) \ \mathbf{else} \ Z(q)) \ \mathbf{else} \ (\mathbf{if} \ y \ \mathbf{then} \ X(q) \ \mathbf{else} \ q)$,
 $\mathbf{CTRL}_1 = \mathbf{if} \ y \ \mathbf{then} \ X(Z(q)) \ \mathbf{else} \ Z(q)$, and $\mathbf{CTRL}_2 = \mathbf{if} \ y \ \mathbf{then} \ X(q) \ \mathbf{else} \ q$.

$$\left[\frac{xy : \mathbf{bool} \otimes \mathbf{bool} \vdash xy : \mathbf{bool} \otimes \mathbf{bool} \quad q : \mathbf{qubit}, x : \mathbf{bool}, y : \mathbf{bool} \vdash U_{XY} : \mathbf{qubit}}{xy : \mathbf{bool} \otimes \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{let} \langle x, y \rangle = xy \mathbf{ in } U_{XY} : \mathbf{qubit}} \right] =$$

$$\left(\{((b_x, b_y), \emptyset) \mapsto [\emptyset]\}, \left(\begin{array}{c} \uparrow \\ \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{((tt,tt),\emptyset)\}} \quad \vdots \quad \left(\begin{array}{c} \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{((tt,ff),\emptyset)\}} \quad \vdots \quad \left(\begin{array}{c} \uparrow \\ \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{((ff,tt),\emptyset)\}} \quad \vdots \quad \left(\begin{array}{c} \uparrow \\ \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{((ff,ff),\emptyset)\}} \right) \quad (4.86)$$

$$\left[\frac{xy : \mathbf{bool} \otimes \mathbf{bool}, q : \mathbf{qubit} \vdash \mathbf{let} \langle x, y \rangle = xy \mathbf{ in } U_{XY} : \mathbf{qubit}}{q : \mathbf{qubit} \vdash \lambda xy. (\mathbf{let} \langle x, y \rangle = xy \mathbf{ in } U_{XY}) : \mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit}} \right] =$$

$$\left(\{ \emptyset \mapsto [\{ (b_x, b_y) \mapsto [\emptyset] \}] \}, \left(\begin{array}{c} \uparrow \\ \text{[Diagram]} \\ \downarrow \\ \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{\emptyset\}} \right) \quad (4.87)$$

$$\left[\frac{q : \mathbf{qubit} \vdash \lambda xy. (\mathbf{let} \langle x, y \rangle = xy \mathbf{ in } U_{XY}) : \mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit}}{\vdash B : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})} \right] =$$

$$\left(\{ \emptyset \mapsto [\{ \emptyset \mapsto [\{ (b_x, b_y) \mapsto [\emptyset] \}] \}] \}, \left(\begin{array}{c} \uparrow \\ \text{[Diagram]} \\ \downarrow \\ \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{\emptyset\}} \right) \quad (4.88)$$

4. Interpretation of the typing derivation of $\vdash \text{Bell} : I \multimap \mathbf{qubit} \otimes \mathbf{qubit}$ where $\text{Bell} = \text{CNOT}(H(\text{init}(tt)), \text{init}(tt))$.

$$\left[\frac{\vdash \text{CNOT} : \mathbf{QQ} \multimap \mathbf{QQ} \quad \vdash \langle H(\text{init}(tt)), \text{init}(tt) \rangle : \mathbf{qubit} \otimes \mathbf{qubit}}{\vdash \text{Bell} : \mathbf{qubit} \otimes \mathbf{qubit}} \right] =$$

$$\left(\{ \emptyset \mapsto [(\emptyset, \emptyset)] \}, \left(\begin{array}{c} \uparrow \\ \text{[Diagram]} \\ \downarrow \end{array} \right)_{\{\emptyset\}} \right) \quad (4.89)$$

where \mathbf{QQ} refers to $\mathbf{qubit} \otimes \mathbf{qubit}$.

5. Interpretation of the typing derivation of $y : \mathbf{qubit}, q : \mathbf{qubit} \vdash \langle A(y), B(q) \rangle :$
 $(\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool}) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})$

$$\left[\frac{\vdash A : \mathbf{qubit} \multimap (\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool}) \quad y : \mathbf{qubit} \vdash y : \mathbf{qubit}}{y : \mathbf{qubit} \vdash A(y) : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})} \right] =$$

$$\left(\left\{ \emptyset \mapsto \left[\left\{ \emptyset \mapsto \left[\begin{array}{l} (tt, tt), (tt, ff), \\ (ff, tt), (ff, ff) \end{array} \right] \right\} \right] \right\}, \left(\begin{array}{c} \text{Circuit diagram for } A(y) \\ \text{with } \{ \emptyset \} \text{ as output} \end{array} \right) \right) \quad (4.90)$$

$$\left[\frac{\vdash B : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit}) \quad q : \mathbf{qubit} \vdash q : \mathbf{qubit}}{q : \mathbf{qubit} \vdash B(q) : (\mathbf{bool} \otimes \mathbf{bool}) \multimap \mathbf{qubit}} \right] =$$

$$\left(\left\{ \emptyset \mapsto \left[\left\{ (b_x, b_y) \mapsto [\emptyset] \mid b_x, b_y \in \{tt, ff\} \right\} \right] \right\}, \left(\begin{array}{c} \text{Circuit diagram for } B(q) \\ \text{with } \{ \emptyset \} \text{ as output} \end{array} \right) \right) \quad (4.91)$$

$$\left[\frac{y : \mathbf{qubit} \vdash A(y) : \mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool}) \quad q : \mathbf{qubit} \vdash B(q) : (\mathbf{bool} \otimes \mathbf{bool}) \multimap \mathbf{qubit}}{y : \mathbf{qubit}, q : \mathbf{qubit} \vdash \langle A(y), B(q) \rangle : (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes ((\mathbf{bool} \otimes \mathbf{bool}) \multimap \mathbf{qubit})} \right] =$$

$$\left(\left\{ \emptyset \mapsto \left[\left(\left\{ \emptyset \mapsto \left[\begin{array}{l} (tt, tt), (tt, ff), \\ (ff, tt), (ff, ff) \end{array} \right] \right\}, \left\{ (b_x, b_y) \mapsto [\emptyset] \mid b_x, b_y \in \{tt, ff\} \right\} \right) \right] \right\}, \left(\begin{array}{c} \text{Circuit diagram for } \langle A(y), B(q) \rangle \\ \text{with } \{ (\emptyset, \emptyset) \} \text{ as output} \end{array} \right) \right) \quad (4.92)$$

6. Interpretation of the typing derivation of $\vdash tel : (\mathbf{qubit} \multimap \mathbf{bool} \otimes \mathbf{bool}) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})$

$$\left[\frac{\vdash Bell : \mathbf{QQ}}{y : \mathbf{qubit}, q : \mathbf{qubit} \vdash \langle A(y), B(q) \rangle : (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit})} \vdash tel : (\mathbf{qubit} \multimap (\mathbf{bool} \otimes \mathbf{bool})) \otimes (\mathbf{bool} \otimes \mathbf{bool} \multimap \mathbf{qubit}) \right] =$$

$$\left\{ \emptyset \mapsto \left[\left(\left\{ \emptyset \mapsto \left[\begin{array}{l} (tt, tt), (tt, ff), \\ (ff, tt), (ff, ff) \end{array} \right] \right\}, \right. \right. \right.$$

$$\left. \left. \left\{ (b_x, b_y) \mapsto [\emptyset] \right\} \right. \right. \left. \left. \mid b_x, b_y \in \{tt, ff\} \right\} \right]$$

$\{\emptyset\}$ (4.93)

where $tel = \mathbf{let} \langle y, q \rangle = Bell(*) \mathbf{in} \langle A(y), B(q) \rangle$.

5 - Soundness

In this chapter, we discuss the soundness of the categorical semantics described in Chapter 4 over the circuit-buffering operational semantics of the quantum channel description language Proto-Quipper-L described in Chapter 3. The soundness of a denotational semantics states that: if two programs M_1 and M_2 of the same type are observationally equivalent, then the denotations of M_1 and M_2 are the same. In this thesis, the observational equivalence of terms is defined by the operational semantics, which describes the reduction of the configuration, and we assume that each term does not contain any classical variable; hence it is a closed term regarding the classical variables.

Our proof is based on the preservation theorem of categorical semantics over the operational semantics. From the theorem, we can prove that if two terms reduce to the value with the same denotation, then all the two terms and the value have the same denotation. Therefore, it suffices to show that when two terms are observationally equivalent (i.e., they reduce to the same value since circuit-buffering operational semantics is terminating by Lemma 3.4.4), then the value must have the same denotation. This leads us to define the basic type, which assures that each value of the type has a unique denotation.

To sum up, our goal in this chapter is to show that if two closed programs M_1 and M_2 of the same basic type reduce to the same value, they have the same denotation.

In the following sections: we, firstly, prove several lemmas; secondly, we define the basic type more formally; and, finally, we provide the proof of the preservation theorem and the soundness theorem of the categorical semantics.

5.1 . Preliminary lemmas

In this section, we provide proofs of lemmas which are required in the following definition of basic types and the proof of soundness theorem of the categorical semantics.

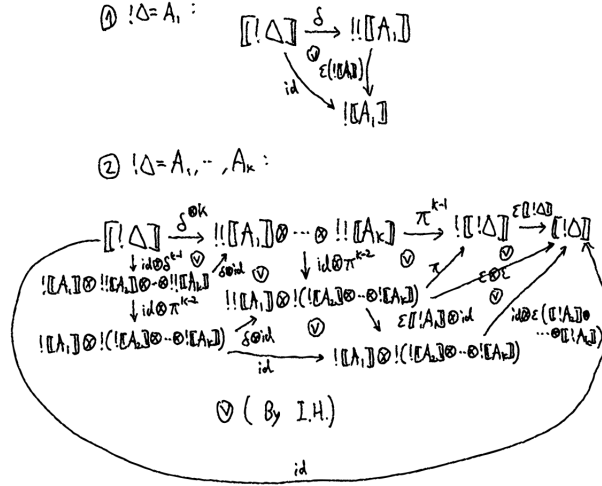
Lemma 5.1.1. *For $\llbracket !\Delta \rrbracket = \llbracket !A_1 \rrbracket \otimes \dots \otimes \llbracket !A_k \rrbracket$, the following diagram commutes:*

$$\begin{array}{ccc}
 \llbracket !\Delta \rrbracket & \xrightarrow{\delta^{\otimes k}} & !! \llbracket A_1 \rrbracket \otimes \dots \otimes !! \llbracket A_k \rrbracket \\
 \downarrow id & & \downarrow \pi^{k-1} \\
 \llbracket !\Delta \rrbracket & \xleftarrow{\epsilon_{\llbracket !\Delta \rrbracket}} & ! \llbracket !\Delta \rrbracket
 \end{array}$$

where $\llbracket !\Delta \rrbracket = ! \llbracket A_1 \rrbracket \otimes \dots \otimes ! \llbracket A_k \rrbracket$.

Proof. Proof by induction on k .

When $! \Delta = \emptyset$, we can show that $\delta^{\otimes 0} = \text{id}_I$ and $\epsilon(I) \circ \pi_I = (\{\emptyset \mapsto (\emptyset, \text{id}_I) \mapsto \emptyset\}, (\text{id}_I)) = \text{id}_I$. Otherwise, it can be shown as follows.



□

Lemma 5.1.2. *The following equation hold:*

$$\left[\frac{\frac{\tau}{! \Delta \vdash V : A} \quad V \text{ is value}}{! \Delta \vdash V : ! A} (p) \mid ! \Delta \vdash V : A \right] = [\tau \mid ! \Delta \vdash V : A]$$

Proof. First, note that

$$\left[\frac{\frac{\tau}{! \Delta \vdash V : A} \quad V \text{ is value}}{! \Delta \vdash V : ! A} (p) \mid ! \Delta \vdash V : A \right] = \frac{[! \Delta] \xrightarrow{\delta^{\otimes k}} \overline{M} !! [A_1] \otimes \dots \otimes !! [A_k] \xrightarrow{\pi^{k-1}} \overline{M} ![! \Delta]}{[\tau \mid ! \Delta \vdash V : A] \xrightarrow{\overline{M}} ! F [A] \xrightarrow{\phi} F ! [A] \xrightarrow{F(\epsilon[A])} \overline{M} F [A]}$$

where $![! \Delta] = ![A_1] \otimes \dots \otimes ![A_k]$. With help of Lemma 4.4.6 and Lemma 5.1.1, we can get the proof in Figure 5.1.

□

5.2 . Basic type

Basic type is a subset of the type (Definition 3.2.1) such that each value of this type has unique and distinct denotation. Note that we are only considering values which do not contain free variables. For example, we can show that the unit type I is basic type since, firstly, there is only two possible values of this

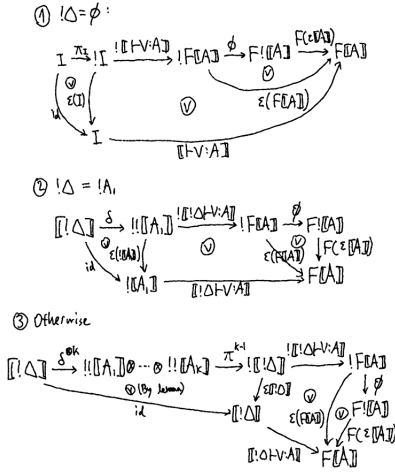


Figure 5.1: Proof of Lemma 5.1.2

type that are variable x and $*$; secondly, there is only one value $*$ which does not contain any free variable; and, thirdly, the value $*$ admits the following forms of typing derivation which has empty context:

$$\frac{}{! \Delta \vdash * : I} (I) \quad \frac{}{! \Delta \vdash * : I} (\pi_*)$$

where π_* represent any type derivation with given premise and conclusion which is composed of promotion (p) and dereliction (d) typing rules. Formally, π_* can be represented as in Eq. (5.1).

$$\pi_* = \frac{}{\alpha_*} (I) \quad | \quad \frac{\alpha_*}{\alpha_*} (\pi_*^2) \quad | \quad \frac{\alpha_* \quad * \text{ is value}}{! \Delta \vdash * : ! I} (p) \quad (5.1) \quad \frac{}{\alpha_*} (\pi_*^1) \quad | \quad \frac{! \Delta \vdash * : ! I}{\alpha_*} (!\pi_*) \quad (d)$$

where $\alpha_* = ! \Delta \vdash * : I$ and $!\pi_I$ and $!\pi_*$ are the type derivation obtained by the same sequence of (p) and (d) type derivation rules but with the types marked by a bang $!$ operator.

Then, we can show by induction that $[[\pi_*]]$ is identity function which maps a morphism to itself since the sequence of rule (p) – (d) can be removed while not changing the denotation as shown in Lemma 5.1.2.

Similarly,

Lemma 5.2.1. *For all typing derivations of $! \Delta \vdash * : I$, their denotation in the categorical semantics is $[[I] \mid ! \Delta \vdash * : I]$.*

Proof. Proof by induction on the size of type derivation. Before, we proceed, there are only three type rules–(d), (p), and (I)–that are applicable to the term $*$.

- basis case where the size of the proof is one: the only type rule which does not contain a premise is (I) rule whose interpretation is $\llbracket (I) \mid !\Delta \vdash * : I \rrbracket$.
- induction step: it is straightforward to see that, for any typing derivation τ of the type judgement $!\Delta \vdash * : I$ whose size is greater than one, there is a application of typing rules (d) followed by (p). Moreover by Lemma 5.1.2, the interpretation of τ is equal to the typing derivation τ' obtained from τ by subtracting the sequence of (d) and (p) rules. Since the size of τ' is smaller, we can apply the induction hypothesis to obtain that

$$\llbracket \tau \mid !\Delta \vdash * : I \rrbracket = \llbracket \tau' \mid !\Delta \vdash * : I \rrbracket = \llbracket (I) \mid !\Delta \vdash * : I \rrbracket$$

□

Lemma 5.2.2. *For all typing derivations of*

$$!\Delta \vdash \text{unbox} : Q\text{Chan}(P, A) \multimap (P \multimap A),$$

their denotation in the categorical semantics is

$$\llbracket (\text{unbox}) \mid !\Delta \vdash \text{unbox} : Q\text{Chan}(P, A) \multimap (P \multimap A) \rrbracket.$$

Proof. From the fact that there are only three typing rules—(unbox), (d), and (p)—which are applicable to the term unbox, we can prove the lemma by a proof by induction similar to Lemma 5.2.1. □

Remark. We can show that I is a basic type but there can be more basic types. For basic type A , there is a denotation of type judgement $\vdash V : A$ for each value V , which does not depend on the typing derivation, and the denotation is distinctive over different values.

5.3 . Soundness theorem of the categorical semantics

In this section, we discuss the soundness theorem of the categorical semantics of the quantum channel description language Proto-Quipper-L. When typed terms admit a unique typing derivation this entails a unique denotation for typed terms. In our situation, due to the promotion and dereliction rules typing derivations are not necessarily unique: we therefore adjust the statements of the lemmas and theorems accordingly. However, in the case of values of basic types, thanks to Remark 5.2 and the type safety properties, the denotation of closed terms of basic types is independent from the choice of typing derivation: this gives the soundness lemma as corollary 5.3.5.1.

First, the value decomposition lemma states that the interpretation of a typing derivation of regarding a value can be decomposed to two parts with a morphism in the category $\overline{\mathcal{M}}$ and the unit of the computation monad.

Lemma 5.3.1. For a value V and a typing derivation τ of the type judgement $!\Delta, Q \vdash V : A$, if $f = \llbracket \tau \mid !\Delta, Q \vdash V : A \rrbracket$ then there exists $f^0 : \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \rightarrow_{\overline{M}} \llbracket A \rrbracket$ such that $f = \eta \circ f^0$.

Moreover, for branching term, if $f = \llbracket \tau \mid \gamma \vdash v : A \rrbracket$ then there exists $f^0 : \llbracket \gamma \rrbracket \rightarrow_{\overline{M}} \llbracket A \rrbracket$ such that $f = \eta \circ f^0$.

Proof. Proof by induction on the typing derivation.

- $\frac{}{!\Delta, (x : A) \vdash x : A}$ (var):

Note that $\llbracket (\text{var}) \mid !\Delta, (x : A) \vdash x : A \rrbracket = \llbracket !\Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{\text{del} \otimes \text{id}}_{\overline{M}} \llbracket I \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{l}_{\overline{M}} \llbracket A \rrbracket \xrightarrow{\eta \llbracket A \rrbracket}}_{\overline{M}} F \llbracket A \rrbracket$. Then we let $f^0 = \llbracket !\Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{\text{del} \otimes \text{id}}_{\overline{M}} \llbracket I \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{l}_{\overline{M}} \llbracket A \rrbracket$ so that we can derive $\llbracket (\text{var}) \mid !\Delta, (x : A) \vdash x : A \rrbracket = \eta \circ f^0$.

Similarly, the cases for the following typing rules can be shown straightforwardly from the definition.

$$\frac{}{!\Delta \vdash * : I} \text{(l)} \quad \frac{}{!\Delta \vdash \text{tt} : \text{bool}} \text{(tt)} \quad \frac{}{!\Delta \vdash \text{ff} : \text{bool}} \text{(ff)}$$

$$\frac{!\Delta, Q, (x : A_a) \vdash M : A_b}{!\Delta, Q \vdash \lambda x. M : A_a \multimap A_b} \text{(-}\circ\text{I)}$$

$$\frac{}{!\Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A)} \text{(box)}$$

$$\frac{}{!\Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A)} \text{(unbox)}$$

$$\frac{p \vDash P \quad \mathbf{vBind}(!\Delta, \text{out}(Q), m, A)}{!\Delta \vdash (p, Q, m) : !\text{QChan}(P, A)} \text{(QChan}_I\text{)}$$

- $\frac{!\Delta, Q \vdash M : !A}{!\Delta, Q \vdash M : A}$ (d):

Note that $\llbracket (\text{d}); \tau \mid !\Delta, Q \vdash M : A \rrbracket = \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{\llbracket !\Delta, Q \vdash M : !A \rrbracket}}_{\overline{M}} F(\llbracket !A \rrbracket) \xrightarrow{F(\epsilon \llbracket A \rrbracket)}}_{\overline{M}} F \llbracket A \rrbracket$. Then, by induction hypothesis, we have that $\llbracket \tau \mid !\Delta, Q \vdash M : !A \rrbracket = \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{f^0}_{\overline{M}} \llbracket !A \rrbracket \xrightarrow{\eta}_{\overline{M}} F(\llbracket !A \rrbracket)$. Therefore, from the following commute diagram, we can conclude that $\llbracket (\text{d}); \tau \mid !\Delta, Q \vdash M : A \rrbracket = \eta \circ \epsilon \llbracket A \rrbracket \circ f^0$.

$$\begin{array}{ccc} \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket & & \\ \downarrow \eta \circ f^0 & \xrightarrow{\epsilon \llbracket A \rrbracket} & \llbracket !A \rrbracket \\ \llbracket !A \rrbracket & & \\ \downarrow \eta & \xrightarrow{F(\epsilon \llbracket A \rrbracket)} & \downarrow \eta \\ F(\llbracket !A \rrbracket) & & F \llbracket A \rrbracket \end{array}$$

- $\frac{!\Delta \vdash V : A \quad V \text{ is value}}{!\Delta \vdash V : !A}$ (p):

Note that $\llbracket (\text{p}); \tau \mid !\Delta \vdash V : !A \rrbracket = \llbracket !\Delta \rrbracket \xrightarrow{\delta \otimes k}_{\overline{M}} \llbracket !A_1 \rrbracket \otimes \dots \otimes \llbracket !A_k \rrbracket \xrightarrow{\pi^{k-1}}_{\overline{M}}$

$! \llbracket !\Delta \rrbracket \xrightarrow{!g}_{\overline{M}} !F \llbracket A \rrbracket \xrightarrow{\phi}_{\overline{M}} F! \llbracket A \rrbracket$, where $\llbracket !\Delta \rrbracket = ! \llbracket A_1 \rrbracket \otimes \dots \otimes ! \llbracket A_k \rrbracket$ and $g = \llbracket \tau \mid !\Delta \vdash V : A \rrbracket$. Then, by induction hypothesis, we have that $g = \llbracket !\Delta \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket A \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket A \rrbracket$. Therefore, from the following commute diagram, we can conclude that $\llbracket (\rho); \tau \mid !\Delta \vdash V : !A \rrbracket = \eta(! \llbracket A \rrbracket) \circ !g^0 \circ \pi^{k-1} \circ \delta^{\otimes k}$.

$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \multimap A_b \quad ! \Delta, Q_b \vdash M_b : A_a}{! \Delta, Q_a, Q_b \vdash M_a M_b : A_b} (\multimap_E):$$

Since the term $M_a M_b$ has to be a value, it follows that $M_a M_b = \text{unbox}(V)$ for some V and, hence, that $A_a = \text{QChan}(P, A)$ and $A_b = (P \multimap A)$. Moreover, according to the term, the typing derivation should have the form

Next, by definition

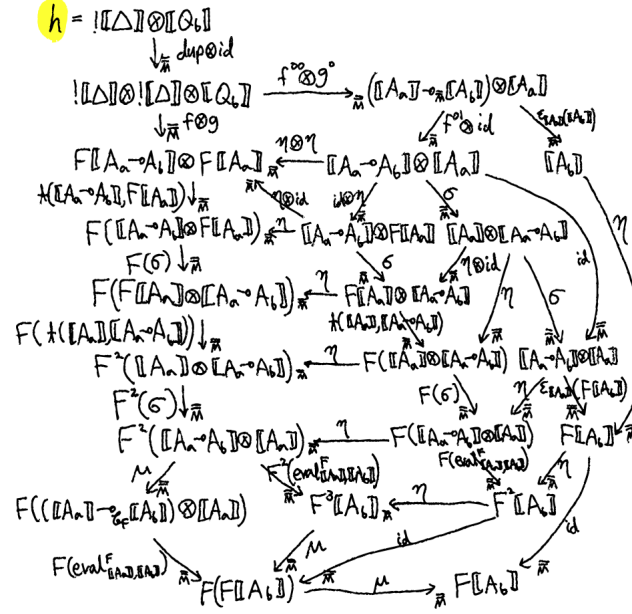
$$\begin{aligned} h &= \llbracket (\multimap_E); (\tau_1, \tau_2) \mid !\Delta, Q_b \vdash M_a M_b : A_b \rrbracket = \\ & \llbracket !\Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup}^{\otimes \text{id}}_{\llbracket Q_b \rrbracket}}_{\overline{M}} \llbracket !\Delta \rrbracket \otimes \llbracket !\Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{f \otimes g}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket \otimes F \llbracket A_a \rrbracket \\ & \xrightarrow{\psi_{\llbracket A_a \multimap A_b \rrbracket, \llbracket A_a \rrbracket}}_{\overline{M}} (F(\llbracket A_a \multimap A_b \rrbracket \otimes \llbracket A_a \rrbracket)) = F(\llbracket A_a \rrbracket \multimap_{\overline{M}_F} \llbracket A_b \rrbracket) \otimes \llbracket A_a \rrbracket \\ & \xrightarrow{F(\text{eval}_{\llbracket A_a \rrbracket, F \llbracket A_b \rrbracket})}_{\overline{M}} F^2 \llbracket A_b \rrbracket \xrightarrow{\mu}_{\overline{M}} F \llbracket A_b \rrbracket \end{aligned}$$

where $f = \llbracket \tau_1 \mid !\Delta \vdash \text{unbox} : A_a \multimap A_b \rrbracket$ and $g = \llbracket \tau_2 \mid !\Delta, Q_b \vdash V : A_a \rrbracket$. Then, by using Lemma 5.2.2, we have that

$$\begin{aligned} f &= \llbracket !\Delta \rrbracket \xrightarrow{\text{del}}_{\overline{M}} I \xrightarrow{\eta_{\llbracket A_a \rrbracket}(I)}_{\overline{M}} \llbracket A_a \rrbracket \multimap (I \otimes \llbracket A_a \rrbracket) \xrightarrow{(\llbracket A_a \rrbracket \multimap -)(\eta \circ \text{unbox})}_{\overline{M}} \llbracket A_a \rrbracket \multimap F \llbracket A_b \rrbracket \\ & \xrightarrow{\eta}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket \\ &= \llbracket !\Delta \rrbracket \xrightarrow{f^{00}}_{\overline{M}} \llbracket A_a \rrbracket \multimap \llbracket A_b \rrbracket \xrightarrow{f^{01} = (\llbracket A_a \rrbracket \multimap -)(\eta)}_{\overline{M}} \llbracket A_a \rrbracket \multimap F \llbracket A_b \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket \end{aligned}$$

Moreover, by induction hypothesis, we obtain that $g = \llbracket !\Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket A_a \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket A_a \rrbracket$. Then, we can obtain that $h = \llbracket !\Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup}^{\otimes \text{id}}_{\llbracket Q_b \rrbracket}}_{\overline{M}} \llbracket !\Delta \rrbracket \otimes \llbracket !\Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{f^{00} \otimes g^0}_{\overline{M}} (\llbracket A_a \rrbracket \multimap \llbracket A_b \rrbracket) \otimes \llbracket A_a \rrbracket \xrightarrow{\text{eval}_{\llbracket A_a \rrbracket, \llbracket A_b \rrbracket}}_{\overline{M}}$

$[[A_b]] \xrightarrow{\eta}_{\overline{M}} F [[A_b]]$ as follows.



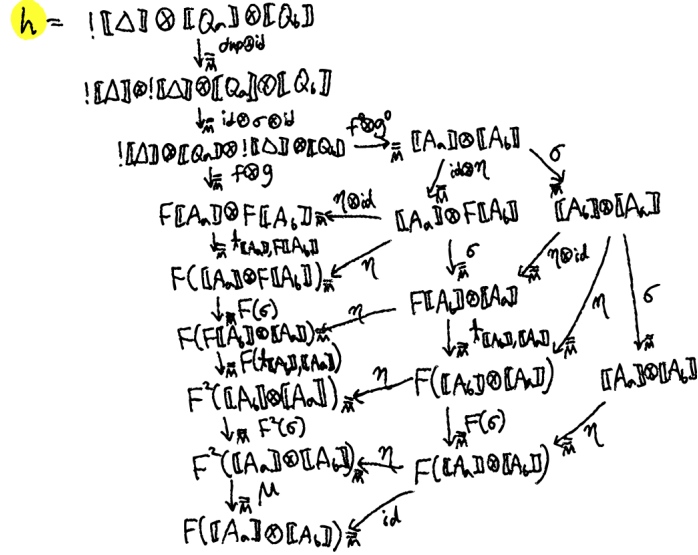
$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \quad ! \Delta, Q_b \vdash M_b : A_b}{! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b} (\otimes_I):$$

By definition, we have the following denotation of the typing rule:

$$\begin{aligned} h &= [(\otimes_I); (\tau_a, \tau_b) \mid ! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b] = \\ & \frac{! \Delta \otimes [Q_a] \otimes [Q_b] \xrightarrow{\text{dup} \otimes \text{id}_{[Q_a] \otimes [Q_b]}}_{\overline{M}} ! \Delta \otimes ! \Delta \otimes [Q_a] \otimes [Q_b]}{\text{id}_{[! \Delta] \otimes \sigma \otimes \text{id}_{[Q_b]}}}_{\overline{M}} ! \Delta \otimes [Q_a] \otimes ! \Delta \otimes [Q_b] \xrightarrow{f \otimes g}_{\overline{M}} F [[A_a]] \otimes F [[A_b]]} \\ & \frac{\psi_{[[A_a], [A_b]]}}{\overline{M}} (F ([[A_a]] \otimes [[A_b]]) = F [[A_a \otimes A_b]]) \end{aligned}$$

where $f = [\tau_a \mid ! \Delta, Q_a \vdash M_a : A_a]$ and $g = [\tau_b \mid ! \Delta, Q_b \vdash M_b : A_b]$. Then, by induction hypothesis, we let $f = ! \Delta \otimes [Q_a] \xrightarrow{f^0}_{\overline{M}} [[A_a]] \xrightarrow{\eta}_{\overline{M}} F [[A_a]]$ and $g = ! \Delta \otimes [Q_b] \xrightarrow{g^0}_{\overline{M}} [[A_b]] \xrightarrow{\eta}_{\overline{M}} F [[A_b]]$. Then, we can obtain that $h = ! \Delta \otimes [Q_a] \otimes [Q_b] \xrightarrow{\text{dup} \otimes \text{id}_{[Q_a] \otimes [Q_b]}}_{\overline{M}} ! \Delta \otimes ! \Delta \otimes [Q_a] \otimes [Q_b] \xrightarrow{\text{id}_{[! \Delta] \otimes \sigma \otimes \text{id}_{[Q_b]}}}_{\overline{M}} ! \Delta \otimes [Q_a] \otimes ! \Delta \otimes [Q_b] \xrightarrow{f^0 \otimes g^0}_{\overline{M}} [[A_a]] \otimes [[A_b]] \xrightarrow{\eta}_{\overline{M}}$

$F(\llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket)$ from the following commute diagram.



- $$\frac{!\Delta, Q_a \vdash M_a : A_a \otimes A_b \quad !\Delta, Q_b, (x : A_a), (y : A_b) \vdash M_b : A}{!\Delta, Q_a, Q_b \vdash \mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b : A} (\otimes_E):$$

This case is irrelevant since $\mathbf{let} \langle x, y \rangle = M_a \mathbf{in} M_b$ is not a value.

- $$\frac{!\Delta, Q_a \vdash M : \mathbf{bool} \quad !\Delta, Q_b \vdash M_a : A \quad !\Delta, Q_b \vdash M_b : A}{!\Delta, Q_a, Q_b \vdash \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b : A} (\mathbf{if}):$$

This case is irrelevant since $\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b$ is not a value.

- $$\frac{\gamma_a \vdash m_a : A \quad \gamma_b \vdash m_b : A}{\gamma_a \times \gamma_b \vdash [m_a, m_b] : A} (\mathbf{b}):$$

The fact that $[m_a, m_b]$ is value implies that both m_a and m_b are values.

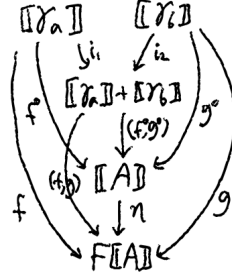
Then, by induction hypothesis, we can represent that $f = \llbracket \tau_a \mid \gamma_a \vdash m_a : A \rrbracket = \llbracket \gamma_a \rrbracket \xrightarrow{f^0} \overline{\llbracket A \rrbracket} \xrightarrow{\eta} \overline{F \llbracket A \rrbracket}$ and $g = \llbracket \tau_b \mid \gamma_b \vdash m_b : A \rrbracket = \llbracket \gamma_b \rrbracket \xrightarrow{g^0} \overline{\llbracket A \rrbracket} \xrightarrow{\eta} \overline{F \llbracket A \rrbracket}$.

Moreover, by definition, we have that $\llbracket (\mathbf{b}); (\tau_a, \tau_b) \mid \gamma_a \times \gamma_b \vdash [m_a, m_b] : A \rrbracket$

is equal to $\llbracket \gamma_a \rrbracket + \llbracket \gamma_b \rrbracket \xrightarrow{(f,g)} \overline{\llbracket A \rrbracket} \xrightarrow{\eta} \overline{F \llbracket A \rrbracket}$. However, we can show that $(f, g) =$

$\llbracket \gamma_a \rrbracket + \llbracket \gamma_b \rrbracket \xrightarrow{(f^0, g^0)} \overline{\llbracket A \rrbracket} \xrightarrow{\eta} \overline{F \llbracket A \rrbracket}$ from the uniqueness of the morphism (f, g) such that $f = (f, g) \circ i_1$ and $g = (f, g) \circ i_2$ since $(f, g) =$

$\eta \circ (f^0, g^0)$ satisfies the condition in the following diagram.



□

Moreover, Lemma 5.3.2 shows that, when the type of a value is non-linear, we can factorize the morphism created by any type derivation of the term.

Lemma 5.3.2. *For any non-branching value V and type derivation τ of the type derivation $\vdash V : !^n B$ where $n \geq 1$ and B is a linear type, then*

$$\llbracket \tau \vdash V : !^n B \rrbracket = I \xrightarrow{f^{(n, f^0)}} \overline{M} !^n B \xrightarrow{\eta} \overline{M} F(!^n B)$$

for some $f^0 : b(I) \rightarrow_{\mathbf{set}} b \llbracket B \rrbracket$, where

$$f^{(n, f^0)} = \begin{cases} I \xrightarrow{\pi_I} \overline{M} ! I \xrightarrow{p^{(f^0)}} \overline{M} ! \llbracket B \rrbracket, & \text{if } n = 1 \\ I \xrightarrow{\pi_I} \overline{M} ! I \xrightarrow{!f^{(n-1, f^0)}} \overline{M} !^n \llbracket B \rrbracket, & \text{otherwise} \end{cases}$$

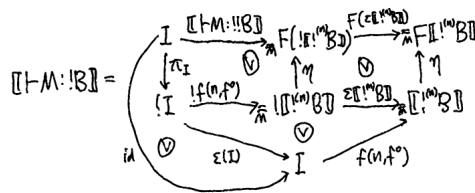
Proof. Proof by induction on the derivation of $\vdash V : !^n B$.

- $\frac{! \Delta, Q \vdash M : !A}{! \Delta, Q \vdash M : A}$ (d) where $! \Delta, Q = \emptyset$, M is a value, and $A = !^n B$:

By induction hypothesis, we have that

$$\begin{aligned} \llbracket \tau \vdash M : !^{n+1} B \rrbracket &= I \xrightarrow{f^{(n+1, f^0)}} \overline{M} !^{n+1} \llbracket B \rrbracket \xrightarrow{\eta} \overline{M} F(!^{n+1} \llbracket B \rrbracket) \\ &= I \xrightarrow{\pi_I} \overline{M} ! I \xrightarrow{!f^{(n, f^0)}} \overline{M} !^{n+1} \llbracket B \rrbracket \xrightarrow{\eta} \overline{M} F(!^{n+1} \llbracket B \rrbracket) \end{aligned}$$

Then: $\llbracket (d); \tau \vdash M : !^n B \rrbracket = I \xrightarrow{\llbracket \tau \vdash M : !^{n+1} B \rrbracket} \overline{M} F(!^{n+1} \llbracket B \rrbracket) \xrightarrow{F(\epsilon \llbracket !^n \llbracket B \rrbracket \rrbracket)} \overline{M} F(!^n \llbracket B \rrbracket)$ by definition and we can obtain the commute diagram required as follows.



- $\frac{! \Delta \vdash V : A \quad V \text{ is value}}{! \Delta \vdash V : !A}$ (p) where $! \Delta = \emptyset$:

We do case analysis:

- if A is a linear type, then by Lemma 5.3.1, we have that

$$\llbracket \tau \vdash V : A \rrbracket = I \xrightarrow{g^0} \overline{M} \llbracket A \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket A \rrbracket$$

and that $\llbracket (\text{p}); \tau \vdash V : !A \rrbracket = I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{! \llbracket \tau \vdash V : A \rrbracket} \overline{M} !F \llbracket A \rrbracket \xrightarrow{\phi} \overline{M} F! \llbracket A \rrbracket$ by definition. Then, we can obtain the required commute diagram as follows.

$$\begin{array}{c} \llbracket \tau \vdash V : !A \rrbracket = I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{!g^0} \overline{M} ! \llbracket A \rrbracket \xrightarrow{! \eta} \overline{M} ! F \llbracket A \rrbracket \\ \downarrow \text{ext} \quad \downarrow \eta \quad \downarrow \phi \\ \llbracket \tau \vdash V : !A \rrbracket = I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{!g^0} \overline{M} ! \llbracket A \rrbracket \xrightarrow{\eta} \overline{M} F! \llbracket A \rrbracket \end{array}$$

- if $A = !^n B$, for $n \geq 1$ and a linear type B , then by induction hypothesis, we have that

$$\llbracket \tau \vdash V : !^n B \rrbracket = I \xrightarrow{f(n, f^0)} \overline{M} \llbracket !^n B \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket !^n B \rrbracket$$

for some f^0 . Then, by definition and the commute diagram shown right above, we can show that

$$\begin{aligned} & \llbracket (\text{p}); \tau \vdash V : !^{n+1} B \rrbracket \\ &= I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{!f(n, f^0)} \overline{M} ! \llbracket !^n B \rrbracket \xrightarrow{! \eta} \overline{M} ! F \llbracket !^n B \rrbracket \xrightarrow{\phi} \overline{M} F! \llbracket !^n B \rrbracket \\ &= I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{!f(n, f^0)} \overline{M} ! \llbracket !^n B \rrbracket \xrightarrow{\eta} \overline{M} F! \llbracket !^n B \rrbracket \\ &= I \xrightarrow{f(n+1, f^0)} \overline{M} \llbracket !^{n+1} B \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket !^{n+1} B \rrbracket \end{aligned}$$

- $\frac{p \vdash P \quad \mathbf{vBind}(! \Delta, \mathbf{out}(Q), m, A)}{! \Delta \vdash (p, Q, m) : ! \text{QChan}(P, A)}$ (QChan_I) where $! \Delta = \emptyset$:

By definition,

$$\begin{aligned} & \llbracket (\text{QChan}_I); (\tau_i)_{i:\text{leaf}} \vdash (p, Q, m) : ! \text{QChan}(P, A) \rrbracket = \\ & I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{!(\eta_{\llbracket P \rrbracket}(I))} \overline{M} !(\llbracket P \rrbracket \multimap \overline{M} (I \otimes \llbracket P \rrbracket)) \xrightarrow{!(\llbracket P \rrbracket \multimap \text{--})(h)} \overline{M} \\ & !(\llbracket P \rrbracket \multimap \overline{M} F \llbracket A \rrbracket) \xrightarrow{\text{box}} \overline{M} ! \llbracket \text{QChan}(P, A) \rrbracket \xrightarrow{\eta} \overline{M} F! \llbracket \text{QChan}(P, A) \rrbracket \end{aligned}$$

for certain morphism h whose construction depends on the typing derivations $(\tau_i)_{i:\text{leaf}}$.

Moreover, since

$$\begin{aligned} \text{box} &= (p \circ b)(\llbracket P \rrbracket \multimap_{\overline{M}} F \llbracket A \rrbracket) \xrightarrow{p(\text{iso}\leftarrow)} p(\overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket)) \\ &\quad \xrightarrow{p(\eta(\overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket)))} (p \circ b \circ p)(\overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket)) \\ &= p(\eta(\overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket))) \circ \text{iso}\leftarrow \end{aligned}$$

we can obtain the required form

$$\begin{aligned} &\llbracket (\text{QChan}_I); (\tau_i)_{i:\text{leaf}} \vdash (p, Q, m) : !\text{QChan}(P, A) \rrbracket \\ &= I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{p(f^0)} \overline{M} !\llbracket \text{QChan}(P, A) \rrbracket \xrightarrow{\eta} \overline{M} F! \llbracket \text{QChan}(P, A) \rrbracket \end{aligned}$$

by letting $f^0 = \eta(\overline{M}(\llbracket P \rrbracket, F \llbracket A \rrbracket))) \circ \text{iso}\leftarrow \circ b((\llbracket P \rrbracket \multimap -)(h)) \circ b(\eta_{\llbracket P \rrbracket}(I))$.

□

Remark. Note that $f^{(n, f^0)} = p(\dots)$ for any $n \geq 1$ and any set function f^0 since $\pi_I = (\{\emptyset \mapsto (\emptyset, \text{id}_I)\}, (\text{id}_I)) = p(\{\emptyset \mapsto (\emptyset, \text{id}_I)\})$.

Regarding the denotation of non-linear value, we can show the following Lemma 5.3.3 which basically represents the naturality property of the inverse of counit (we may call it pseudo unit) from the !-comonad.

Lemma 5.3.3. *The following diagram commutes:*

$$\begin{array}{ccc} I \otimes I & \xrightarrow{\text{id} \otimes g^0} & I \otimes !\llbracket B' \rrbracket \\ \downarrow l & & \downarrow l \\ I & \xrightarrow{g^0} & !\llbracket B' \rrbracket \\ \downarrow \pi_I & & \downarrow \delta \\ !I & \xrightarrow{!g^0} & !\llbracket B \rrbracket \end{array}$$

where $g^0 = f^{(n, g_0^0)}$.

Proof. Proof by case analysis on n .

- $n = 1$: By definition, we have that $g^0 = I \xrightarrow{\pi_I} \overline{M} !I \xrightarrow{p(g_0^0)} \overline{M} !\llbracket B' \rrbracket$.

Then we can show the following commute diagram:

$$\begin{array}{ccccc} I & \xrightarrow{\pi_I} & !I & \xrightarrow{p(g_0^0)} & !\llbracket B' \rrbracket \\ \downarrow \pi_I & & \downarrow p(\eta(b(I))) & & \delta \downarrow \\ !I & \xrightarrow{!\pi_I} & !!I & \xrightarrow{!p(g_0^0)} & !!\llbracket B' \rrbracket \end{array}$$

Then we can obtain the right part of the commute diagram by using the fact that $\delta = p \circ \eta \circ b$ and the naturality of $\eta : \mathbf{1}_{\mathbf{Set}} \rightarrow (b \circ p)$:

$$\begin{array}{ccc} b(I) & \xrightarrow{g_0^0} & b \llbracket B' \rrbracket \\ \downarrow \eta(b(I)) & & \eta(b \llbracket B' \rrbracket) \downarrow \\ (b \circ p \circ b)(I) & \xrightarrow{(b \circ p)(g_0^0)} & (b \circ p \circ b) \llbracket B' \rrbracket \end{array}$$

Moreover, the left part of the commute diagram can be shown by using the fact that $I = p(\{\emptyset\})$, $\pi_I = p(\eta(\{\emptyset\}))$, and the naturality of $\eta : \mathbf{1}_{\mathbf{Set}} \rightarrow (b \circ p)$:

$$\begin{array}{ccc} \{\emptyset\} & \xrightarrow{\eta(\{\emptyset\})} & (b \circ p)(\{\emptyset\}) & & p(\{\emptyset\}) & \xrightarrow{p(\eta(\{\emptyset\}))} & (p \circ b \circ p)(\{\emptyset\}) \\ \downarrow \eta(\{\emptyset\}) & & \downarrow \eta((b \circ p)(\{\emptyset\})) & \implies & \downarrow p(\eta(\{\emptyset\})) & & \downarrow p(\eta((b \circ p)(\{\emptyset\}))) \\ (b \circ p)(\{\emptyset\}) & \xrightarrow{(b \circ p)(\eta(\{\emptyset\}))} & (b \circ p \circ b \circ b)(\{\emptyset\}) & & (p \circ b \circ p)(\{\emptyset\}) & \xrightarrow{(p \circ b \circ p)(\eta(\{\emptyset\}))} & (p \circ b \circ p \circ b \circ b)(\{\emptyset\}) \end{array}$$

- $n > 1$: By definition we have that $g^0 = I \xrightarrow{\pi_I} \overline{!I} \xrightarrow{!f^{(n-1), g_0^0}} \overline{! \llbracket B' \rrbracket}$.

Then we can show the following commute diagram:

$$\begin{array}{ccc} I & \xrightarrow{\pi_I} & !I & \xrightarrow{!f^{(n-1), g_0^0}} & ! \llbracket B' \rrbracket \\ \downarrow \pi_I & & \downarrow \delta(I) & & \delta \llbracket B' \rrbracket \downarrow \\ !I & \xrightarrow{! \pi_I} & !!I & \xrightarrow{!!f^{(n-1), g_0^0}} & !! \llbracket B' \rrbracket \end{array}$$

The left part of the commute diagram can be shown as the previous case and the right part follows from the naturality of $\delta = p \circ \eta \circ b$.

□

Next, the substitution lemma (Lemma 5.3.4) provides the interpretation of the substitution in a term. It consists of two parts: one for the typing judgement and the other for the νBind . Intuitively, for the typing judgement, the lemma says that the substitution of a variable in a term by value corresponds to the composition of the morphisms for the value and the term.

Lemma 5.3.4. *The following holds,*

- *for typing judgement:*
Given a typing derivation τ_f for the type judgement $Q_1, (x : B) \vdash M : A$ and the interpretation $f = \llbracket \tau_f \mid Q_1, (x : B) \vdash M : A \rrbracket$ and a typing derivation τ_g for the type judgement $Q_2 \vdash V : B$ and the interpretation $g =$

$\llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket$, there exists a type derivation τ_{vb} of $Q_1, Q_2 \vdash M[V/x] : A$ whose denotation $f_{vb} = \llbracket \tau_{vb} \mid Q_1, Q_2 \vdash M[V/x] : A \rrbracket$ satisfies the following commute diagram in the category \overline{M} :

$$\begin{array}{ccc} \llbracket Q_1 \rrbracket \otimes \llbracket Q_2 \rrbracket & \xrightarrow{id \otimes g^0} & \llbracket Q_1 \rrbracket \otimes \llbracket B \rrbracket \\ & \searrow \exists f_{vb} & \downarrow f \\ & & F \llbracket A \rrbracket \end{array}$$

where $g = \eta \circ g^0$ by the value decomposition lemma.

• for **vBind**:

Given any derivation τ_g of the judgement **vBind** $((x : !A'), c, m[V/x], A)$ and any typing derivation τ_v of the typing judgement $\vdash V : !A'$ for a value V , there exists a derivation τ_{vb} of **vBind** $(\emptyset, c, m[V/x], A)$ whose denotation $f_{vb} = \llbracket \tau_{vb} \mid \mathbf{vBind}(\emptyset, c, m[V/x], A) \rrbracket$ satisfies the following commute diagram:

$$\begin{array}{ccc} \llbracket c \rrbracket = \llbracket c \rrbracket \otimes I & \xrightarrow{id \otimes v^0} & \llbracket c \rrbracket \otimes \llbracket !A' \rrbracket \\ \downarrow \exists f_{vb} & & \downarrow \sigma \\ F \llbracket A \rrbracket & \xleftarrow{g} & \llbracket !A' \rrbracket \otimes \llbracket c \rrbracket \end{array}$$

where $g = \llbracket \tau_g \mid \mathbf{vBind}((x : !A'), c, m, A) \rrbracket$ and $\llbracket \tau_v \mid \vdash V : !A' \rrbracket = \eta \circ v^0$.

Proof. The proof consists of two parts.

Typing judgement part Proof by induction on the type derivation τ_f of $Q_1, (x : B) \vdash M : A$.

• $\frac{}{! \Delta, (x : A) \vdash x : A}$ (var):

First, note that $f = \llbracket (\text{var}) \mid ! \Delta, (x : A) \vdash x : A \rrbracket = \llbracket ! \Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{\text{del} \otimes \text{id}} \overline{M} I \otimes \llbracket A \rrbracket \xrightarrow{l} \overline{M} \llbracket A \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket A \rrbracket$ and $g = \llbracket ! \Delta \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{g^0} \overline{M} \llbracket A \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket A \rrbracket$ by Lemma 5.3.1 where $! \Delta = \emptyset$ and $Q_1 = \emptyset$.

There are two cases, either $(M = x)$ and, hence, $M[V/x] = V$ or $(M = y)$, hence, $M[V/x] = M$. If the substitution happens, i.e. $M[V/x] = V$, we can show that the same type derivation τ_g from g gives us the witness, the morphism $g = \llbracket \tau_g \mid ! \Delta, Q_2 \vdash V : A \rrbracket$, of the desired commute diagram as follows.

$$\begin{array}{ccc} \llbracket \emptyset \rrbracket \otimes \llbracket Q_2 \rrbracket & \xrightarrow{id \otimes g^0} & \llbracket \emptyset \rrbracket \otimes \llbracket A \rrbracket \\ \downarrow \llbracket \tau_g \mid Q_2 \vdash V : A \rrbracket & \searrow g^0 & \downarrow \text{del} \otimes \text{id} = \text{id} \\ \llbracket \emptyset \rrbracket \otimes \llbracket A \rrbracket & & \llbracket \emptyset \rrbracket \otimes \llbracket A \rrbracket \\ \downarrow \eta & \swarrow \eta & \downarrow \text{id} \\ F \llbracket A \rrbracket & \xleftarrow{g} & \llbracket A \rrbracket \end{array}$$

Otherwise, $(M = y)$ and, hence, to be able to apply (var) rule to prove the typing judgement $Q_1, (x : B) \vdash M : A$, the type $(y : A)$ should appear in the context Q_1 and $(x : B)$ belongs to the non-linear context $!\Delta$, which is supposed to be empty. Therefore, this case is absurde.

- $\frac{!\Delta, Q \vdash M : !A}{!\Delta, Q \vdash M : A}$ (d):

Since the typing derivation τ_f is obtained by applying (d)-rule, we know that the type judgement $Q_1, (x : B) \vdash M : A$ has the form $!\Delta, Q \vdash M : A$. Therefore, we let the context $!\Delta, Q$ be $Q_1, (x : B)$.

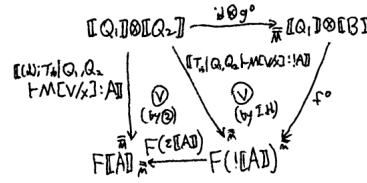
By definition, we have that

$$f = \llbracket (d); \tau_f \mid !\Delta, Q \vdash M : A \rrbracket = \llbracket !\Delta \rrbracket \otimes \llbracket Q \rrbracket \xrightarrow{f^0} \overline{M} F(![A]) \xrightarrow{F(\epsilon[A])} \overline{M} F[A],$$

where

$$f^0 = \llbracket \tau_f \mid !\Delta, Q \vdash M : !A \rrbracket \text{ and } g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0} \overline{M} [B] \xrightarrow{\eta} \overline{M} F[B]$$

by Lemma 5.3.1. Then, we can derive the commute diagram as follows.



where we have the followings:

1. by induction hypothesis, there exists some derivation τ_{vb} such that

$$\llbracket \tau_{vb} \mid Q_1, Q_2 \vdash M[V/x] : !A \rrbracket = \llbracket Q_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{id \otimes g^0} \overline{M} [!\Delta'] \otimes \llbracket Q_1 \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{f^0} \overline{M} F(![A])$$

2. from the type derivation

$$\frac{Q_1, Q_2 \vdash M[V/x] : !A}{Q_1, Q_2 \vdash M[V/x] : A}$$
(d)

we get

$$\llbracket (d); \tau_{vb} \mid Q_1, Q_2 \vdash M[V/x] : A \rrbracket = \llbracket Q_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\llbracket \tau_{vb} \mid Q_1, Q_2 \vdash M[V/x] : !A \rrbracket} \overline{M} F(![A]) \xrightarrow{F(\epsilon[A])} \overline{M} F[A]$$

- $\frac{! \Delta \vdash V : A \quad V \text{ is value}}{! \Delta \vdash V : !A}$ (p):

Since the typing derivation τ_f is obtained by applying (p)-rule, we know that the type judgement $Q_1, (x : B) \vdash M : A$ has the form $! \Delta \vdash V : !A$, which implies that $Q_1 = \emptyset$ and $B = !B'$ for some B' . Moreover, we know that the term M is a value V .

Then, by definition,

$$f = \llbracket (p); \tau_f \mid (x : B) \vdash V : !A \rrbracket = \llbracket !B' \rrbracket \xrightarrow{\delta}_{\overline{M}} !! \llbracket B' \rrbracket \xrightarrow{!(f^0)}_{\overline{M}} !F \llbracket A \rrbracket \xrightarrow{\phi}_{\overline{M}} F! \llbracket A \rrbracket$$

and $g = \llbracket \tau_g \mid Q_2 \vdash V' : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket B \rrbracket$ where $f^0 = \llbracket \tau_f \mid (x : B) \vdash V : A \rrbracket$ and $g^0 = f^{(n, g^0)}$ by Lemma 5.3.2 since V' is value. Moreover, note that, by Lemma 3.2.4, we know that $Q_2 = \emptyset$ since the type $B = !B'$ in the type judgement $Q_2 \vdash V' : B$ is non-linear.

Now we have the followings:

1. By induction hypothesis, we know that there exists some typing derivation τ_{vb} of the type judgement $\vdash V[V'/x] : A$ whose denotation satisfies the following equality:

$$\llbracket \tau_{vb} \mid \vdash V[V'/x] : A \rrbracket = I \xrightarrow{g^0}_{\overline{M}} \llbracket B \rrbracket \xrightarrow{f^0}_{\overline{M}} F \llbracket A \rrbracket$$

2. From Lemma 3.1.6, we know that $V[V'/x]$ is value. Therefore, we have the following type derivation:

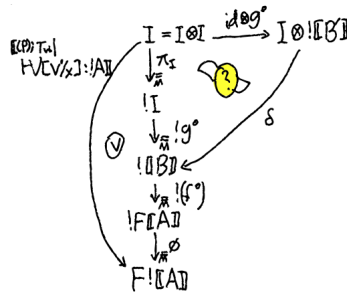
$$\frac{\vdash V[V'/x] : A \quad V[V'/x] \text{ is value}}{\vdash V[V'/x] : !A} \text{ (p)}$$

whose denotation is given by:

$$\begin{aligned} \llbracket (p); \tau_{vb} \mid \vdash V[V'/x] : !A \rrbracket &= I \xrightarrow{\pi_I}_{\overline{M}} !I \xrightarrow{!\llbracket \tau_{vb} \mid \vdash V[V'/x] : A \rrbracket}_{\overline{M}} !F \llbracket A \rrbracket \xrightarrow{\phi}_{\overline{M}} F! \llbracket A \rrbracket \\ &= I \xrightarrow{\pi_I}_{\overline{M}} !I \xrightarrow{!g^0}_{\overline{M}} !\llbracket B \rrbracket \xrightarrow{!f^0}_{\overline{M}} !F \llbracket A \rrbracket \xrightarrow{\phi}_{\overline{M}} F! \llbracket A \rrbracket \end{aligned}$$

from the induction hypothesis.

Then, it suffices to show that our typing derivation (p); τ_{vb} satisfies the desired commute diagram as follows.



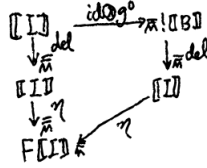
where the commute diagram in the middle follows from Lemma 5.3.3.

- $\frac{}{! \Delta \vdash * : I} (l)$:

This is the case where the typing derivation τ_f of the type judgement $Q_1, (x : B) \vdash M : A$ is derived by applying (l)-rule. Therefore, we let $! \Delta = Q_1, (x : B)$, which implies that $Q_1 = \emptyset$ and $B = !B'$ for some B' , and $M = *$.

Moreover, by definition, we have that $f = \llbracket (l) \mid (x : B) \vdash * : I \rrbracket = \llbracket !B' \rrbracket \xrightarrow{\text{del}}_{\overline{M}} I \xrightarrow{\eta}_{\overline{M}} F \llbracket I \rrbracket$, and, by Lemma 5.3.1, $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0}_{\overline{M}} ! \llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F (! \llbracket B \rrbracket)$ where we know that $Q_2 = \emptyset$ by Lemma 3.2.4 since $B = !B'$ is non-linear. Also, since B is non-linear, by Lemma 5.3.2, we know that $g^0 = f^{(n, g_0^0)}$ for some n and morphism g_0^0 , which implies that $g^0 = p(\dots)$.

Then, we have that $*[V/x] = *$ and that $\llbracket (l) \mid *[V/x] : I \rrbracket = \llbracket !B' \rrbracket \xrightarrow{\text{del}}_{\overline{M}} \llbracket I \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket I \rrbracket$. Therefore, by letting $\tau_{vb} = (l)$, the desired commute diagram as follows.



where the commute diagram in the middle follows from Lemma 4.5.4 since $\llbracket !\Delta' \rrbracket = I$ is a parameter object and the morphisms dup and g^0 are morphisms of parameter objects of the form $p(\dots)$.

- $\frac{! \Delta, Q, (y : A_a) \vdash M' : A_b}{! \Delta, Q \vdash \lambda y. M' : A_a \multimap A_b} (\multimap_I)$:

Suppose that the typing derivation of the type judgement $Q_1, (x : B) \vdash M : A$ is $(\multimap_I); \tau_f$. Then, we know that $! \Delta, Q = Q'_1, (x : B)$ and M by $\lambda y. M'$.

By definition, we have that $f = \llbracket (\multimap_I); \tau_f \mid ! \Delta, Q \vdash \lambda y. M' : A_a \multimap A_b \rrbracket$ is equal to

$$\llbracket ! \Delta, Q \rrbracket \xrightarrow{\eta_{\llbracket A_a \rrbracket}(\llbracket ! \Delta, Q \rrbracket)}_{\overline{M}} \llbracket A_a \rrbracket \multimap (\llbracket ! \Delta, Q \rrbracket \otimes \llbracket A_a \rrbracket) \xrightarrow{(\llbracket A_a \rrbracket \multimap -)(f^0)}_{\overline{M}} \llbracket A_a \rrbracket \multimap F \llbracket A_b \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket,$$

where $f^0 = \llbracket \tau_f \mid ! \Delta, Q, (y : A_a) \vdash M' : A_b \rrbracket$. By Lemma 5.3.1, we have that

$$g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket B \rrbracket.$$

Next, we have the following facts:

1. By induction hypothesis, we know that there is a typing derivation τ_{vb} for the type judgement $Q'_1, (y : A_a), Q_2 \vdash M'[V/x] : A_b$ which is:

$$\begin{aligned} & \llbracket \tau_{vb} \mid Q'_1, (y : A_a), Q_2 \vdash M'[V/x] : A_b \rrbracket = \\ & \llbracket Q'_1 \rrbracket \otimes \llbracket A_a \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0} \llbracket Q'_1 \rrbracket \otimes \llbracket A_a \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{\text{id} \otimes \sigma} \llbracket M \rrbracket \\ & \llbracket Q'_1 \rrbracket \otimes \llbracket B \rrbracket \otimes \llbracket A_a \rrbracket \xrightarrow{f^0} \llbracket M \rrbracket F \llbracket A_b \rrbracket \end{aligned}$$

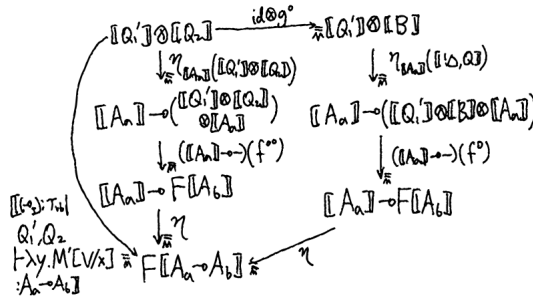
2. The fact that $\lambda y.M'[V/x] = (\lambda y.M')[V/x]$ and the interpretation of the typing rule:

$$\frac{Q'_1, Q_2, (y : A_a) \vdash M'[V/x] : A_b}{Q'_1, Q_2 \vdash \lambda y.M'[V/x] : A_a \multimap A_b} (\multimap_I)$$

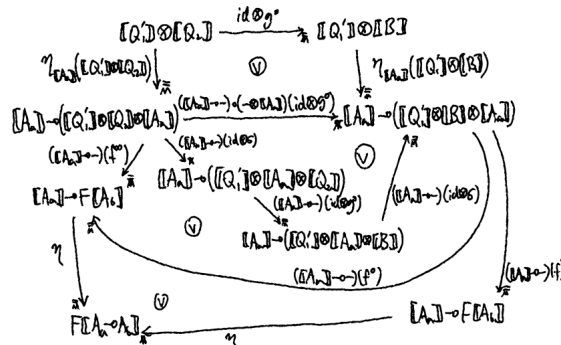
$$\begin{aligned} & \llbracket (\multimap_I); \tau_{vb} \mid Q'_1, Q_2 \vdash \lambda y.M'[V/x] : A_a \multimap A_b \rrbracket = \\ & \llbracket Q'_1, Q_2 \rrbracket \xrightarrow{\eta_{A_a}(\llbracket Q'_1, Q_2 \rrbracket)} \llbracket A_a \rrbracket \multimap (\llbracket Q'_1, Q_2 \rrbracket \otimes \llbracket A_a \rrbracket) \xrightarrow{(\llbracket A_a \rrbracket \multimap \multimap)(f^{00})} \llbracket M \rrbracket \\ & \llbracket A_a \rrbracket \multimap F \llbracket A_b \rrbracket \xrightarrow{\eta} \llbracket M \rrbracket F \llbracket A_a \multimap A_b \rrbracket \end{aligned}$$

where $f^{00} = (\text{id} \otimes \sigma)$; $\llbracket \tau_{vb} \mid Q'_1, (y : A_a), Q_2 \vdash M'[V/x] : A_b \rrbracket$.

Therefore, we can show that the typing derivation $(\multimap_I); \tau_{vb}$ proves the desired commute diagram as follows:



Then, it suffices to show the following commute diagram as follows:



$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \multimap A_b \quad ! \Delta, Q_b \vdash M_b : A_a}{! \Delta, Q_a, Q_b \vdash M_a M_b : A_b} (\multimap E);$$

Suppose that the typing derivation of the type judgement $Q_1, (x : B) \vdash M : A$ is $(\multimap E); (\tau_1, \tau_2)$. Then, we know that $! \Delta, Q_a, Q_b = Q'_1, (x : B)$.

By definition, we have that $f' = \llbracket (\multimap E); (\tau_1, \tau_2) \mid ! \Delta, Q_a, Q_b \vdash M_a M_b : A_b \rrbracket = \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup} \otimes \text{id}}_{\overline{M}} \llbracket ! \Delta \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{id} \otimes \sigma \otimes \text{id}}_{\overline{M}} \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^1 \otimes h^2}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket \otimes F \llbracket A_a \rrbracket \xrightarrow{\psi}_{\overline{M}} F \llbracket (A_a \multimap A_b) \otimes A_a \rrbracket \xrightarrow{F(\text{eval}_{\llbracket A_a \rrbracket, F \llbracket A_b \rrbracket})}}_{\overline{M}} F^2 \llbracket A_b \rrbracket \xrightarrow{\mu}_{\overline{M}} F \llbracket A_b \rrbracket$ where $h^1 = \llbracket \tau_1 \mid ! \Delta, Q_a \vdash M_a : A_a \multimap A_b \rrbracket$ and $h^2 = \llbracket \tau_2 \mid ! \Delta, Q_b \vdash M_b : A_a \rrbracket$.

For convenience, we define the morphism $f = f' \circ \sigma_f : \llbracket Q'_1 \rrbracket \otimes \llbracket B \rrbracket \rightarrow_{\overline{M}} F \llbracket A_b \rrbracket$ which changes the order of the variables in the context.

By Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket B \rrbracket$.

The goal is then to show that there exists a typing derivation τ'_{vb} of the type judgement $Q'_1, Q_2 \vdash M_a M_b[V/x] : A_b$ which satisfies that

$$\llbracket \tau'_{vb} \mid Q'_1, Q_2 \vdash M_a M_b[V/x] : A_b \rrbracket = \llbracket Q'_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket Q'_1 \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{f}_{\overline{M}} F \llbracket A_b \rrbracket.$$

To prove that, we analyze the following three cases:

- $(x : B) \in ! \Delta$, which means that $B = !B'$, $! \Delta = (x : !B')$, and $Q'_1 = Q_a, Q_b$:

Note that $Q_2 = \emptyset$ is deducible from $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket$ by Lemma 3.2.4 since $B = !B'$ is non-linear.

1. Induction hypotheses: there are typing derivations τ_{vb}^a for the type judgement $Q_a \vdash M_a[V/x] : A_a \multimap A_b$ and τ_{vb}^b for $Q_b \vdash M_b[V/x] : A_a$ such that

$$\begin{aligned} h_a &= \llbracket \tau_{vb}^a \mid Q_a \vdash M_a[V/x] : A_a \multimap A_b \rrbracket \\ &= \llbracket Q_a \rrbracket \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket Q_a \rrbracket \otimes ! \llbracket B' \rrbracket \xrightarrow{\sigma}_{\overline{M}} ! \llbracket B' \rrbracket \otimes \llbracket Q_a \rrbracket \xrightarrow{h^1}_{\overline{M}} F \llbracket A_a \multimap A_b \rrbracket \\ h_b &= \llbracket \tau_{vb}^b \mid Q_b \vdash M_b[V/x] : A_a \rrbracket \\ &= \llbracket Q_b \rrbracket \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket Q_b \rrbracket \otimes ! \Delta B' \xrightarrow{\sigma}_{\overline{M}} ! \llbracket B' \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^2}_{\overline{M}} F \llbracket A_a \rrbracket \end{aligned}$$

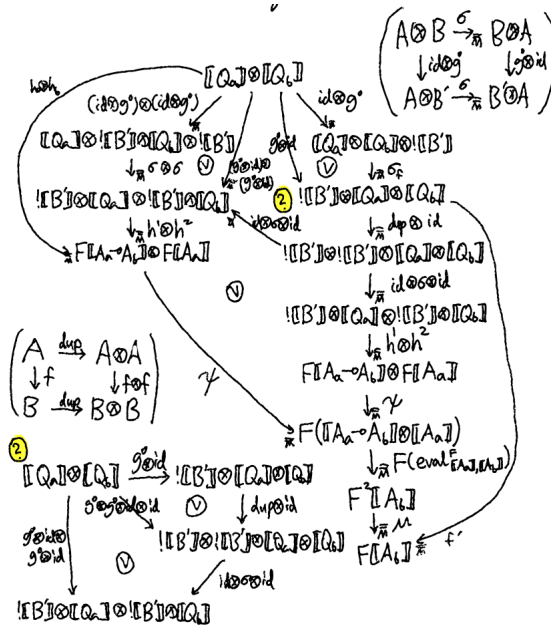
2. We have the following typing derivation:

$$\frac{Q_a \vdash M_a[V/x] : A_a \multimap A_b \quad Q_b \vdash M_b[V/x] : A_a}{Q'_1 \vdash (M_a M_b)[V/x] : A_b} (\multimap E)$$

whose denotation is given by

$$\begin{aligned} & \left[(\multimap E); (\tau_{vb}^a, \tau_{vb}^b) \mid Q_1 \vdash (M_a M_b)[V/x] : A_b \right] = \\ & \left[[Q_a] \otimes [Q_b] \xrightarrow{h_a \otimes h_b} \overline{M} F[A_a \multimap A_b] \otimes F[A_a] \xrightarrow{\psi} \overline{M} \right. \\ & \left. F([A_a \multimap A_b] \otimes [A_a]) \xrightarrow{\epsilon_{[A_a]}(F[A_b])} \overline{M} F^2[A_b] \xrightarrow{\mu} \overline{M} F[A_b] \right] \end{aligned}$$

Given these facts, we can derive the goal from following diagram.



- $(x : B) \in Q_a$, which means that $!\Delta = \emptyset$, $Q_a = Q'_a, (x : B)$, and $Q'_1 = Q'_a \otimes Q_b$:

In this case, $M_b[V/x] = M_b$ since $\mathbf{FV}(M_b) \subseteq \mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_b)$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^a for the type judgement $Q'_a, Q_2 \vdash M_a[V/x] : A_a \multimap A_b$ such that

$$\begin{aligned} & \left[\tau_{vb}^a \mid Q'_a, Q_2 \vdash M_a[V/x] : A_a \multimap A_b \right] \\ & = \left[[Q'_a] \otimes [Q_2] \xrightarrow{\text{id} \otimes g^0} \overline{M} [Q'_a] \otimes [B] \xrightarrow{h^1} \overline{M} F[A_a \multimap A_b] \right] \end{aligned}$$

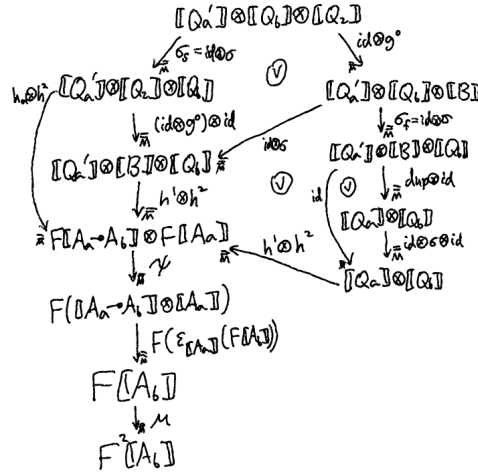
2. We have the following typing derivation:

$$\frac{Q'_a, Q_2 \vdash M_a[V/x] : A_a \multimap A_b \quad Q_b \vdash M_b : A_a}{Q'_a, Q_2, Q_b \vdash M_a M_b[V/x] : A_b} (\multimap E)$$

whose denotation is

$$\begin{aligned}
& \llbracket (\neg \circ E); (\tau_{vb}^a, \tau_2) \mid Q'_a, Q_b, Q_2 \vdash (M_a M_b)[V/x] : A_b \rrbracket \\
&= \llbracket (\neg \circ E); (\tau_{vb}^a, \tau_2) \mid Q'_a, Q_2, Q_b \vdash (M_a M_b)[V/x] : A_b \rrbracket \circ \sigma_s \\
&= \llbracket Q'_a \rrbracket \otimes \llbracket Q_b \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes \sigma} \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \otimes \llbracket Q_b \rrbracket \\
&\xrightarrow{h_a \otimes h^2} \llbracket F[A_a \multimap A_b] \rrbracket \otimes \llbracket F[A_a] \rrbracket \xrightarrow{\psi} \llbracket F([A_a \multimap A_b]) \otimes [A_a] \rrbracket \\
&\xrightarrow{F(\epsilon_{[A_a]}(F[A_b]))} \llbracket F^2[A_b] \rrbracket \xrightarrow{\mu} \llbracket F[A_b] \rrbracket
\end{aligned}$$

From these facts, the goal can be derived from the following diagram.



- $(x : B) \in Q_b$, which means that $!\Delta = \emptyset$, $Q_b = Q'_b, (x : B)$, and $Q'_1 = Q_a, Q'_b$:

In this case, $M_a[V/x] = M_a$ since $\mathbf{FV}(M_a) \subseteq \mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_a)$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^b for the type judgement $Q'_b, Q_2 \vdash M_b[V/x] : A_a$ such that

$$\begin{aligned}
& \llbracket \tau_{vb}^b \mid Q'_b, Q_2 \vdash M_b[V/x] : A_a \rrbracket \\
&= \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0} \llbracket Q'_b \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{h^2} \llbracket F[A_a] \rrbracket
\end{aligned}$$

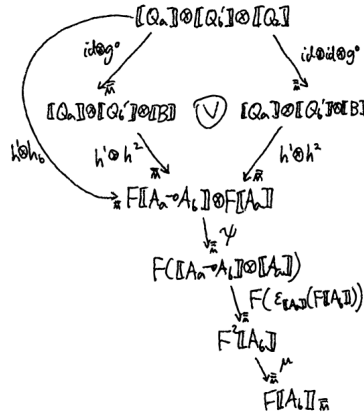
2. We have the following typing derivation:

$$\frac{Q_a \vdash M_a : A_a \multimap A_b \quad Q'_b, Q_2 \vdash M_b[V/x] : A_a}{Q_a, Q'_b, Q_2 \vdash (M_a M_b)[V/x] : A_b} (\neg \circ E)$$

whose denotation is

$$\begin{aligned}
& \left[(\neg \circ_E); (\tau_1, \tau_{vb}^b) \mid Q_a, Q'_b, Q_2 \vdash (M_a M_b)[V/x] : A_b \right] \\
&= \llbracket Q_a \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{h^1 \otimes h_b} \overline{M} F \llbracket A_a \neg \circ A_b \rrbracket \otimes F \llbracket A_a \rrbracket \\
&\xrightarrow{\psi} \overline{M} F(\llbracket A_a \neg \circ A_b \rrbracket \otimes \llbracket A_a \rrbracket) \xrightarrow{F(\epsilon_{\llbracket A_a \rrbracket}(F \llbracket A_b \rrbracket))} \overline{M} F^2 \llbracket A_b \rrbracket \xrightarrow{\mu} \overline{M} F \llbracket A_b \rrbracket
\end{aligned}$$

From these facts, the goal can be derived from the following diagram.



$$\bullet \frac{! \Delta, Q_a \vdash M_a : A_a \quad ! \Delta, Q_b \vdash M_b : A_b}{! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b} (\otimes_I):$$

Suppose that the typing derivation of $Q_1, (x : B) \vdash M : A$ is obtained by applying (\otimes_I) -rule. Then, we know that $! \Delta, Q_a, Q_b = Q_1, (x : B)$ and $M = \langle M_a, M_b \rangle$.

By definition, we have that

$$\begin{aligned}
f' &= \llbracket (\otimes_I); (\tau_1, \tau_2) \mid ! \Delta, Q_a, Q_b \vdash \langle M_a, M_b \rangle : A_a \otimes A_b \rrbracket \\
&= \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup} \otimes \text{id}} \overline{M} \llbracket ! \Delta \rrbracket \otimes \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \\
&\xrightarrow{\text{id} \otimes \sigma \otimes \text{id}} \overline{M} \llbracket ! \Delta \rrbracket \otimes \llbracket Q_a \rrbracket \llbracket ! \Delta \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^1 \otimes h^2} \overline{M} \\
&F \llbracket A_a \rrbracket \otimes F \llbracket A_b \rrbracket \xrightarrow{\psi} \overline{M} F(\llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket)
\end{aligned}$$

where $h^1 = \llbracket \tau_1 \mid ! \Delta, Q_a \vdash M_a : A_a \rrbracket$ and $h^2 = \llbracket \tau_2 \mid ! \Delta, Q_b \vdash M_b : A_b \rrbracket$.

For convenience, we define the morphism $f = f' \circ \sigma_f : \llbracket Q'_a \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket B \rrbracket \rightarrow \overline{M} F \llbracket A_a \otimes A_b \rrbracket$ which changes the order of the variables in the context.

By Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0} \overline{M} \llbracket B \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket B \rrbracket$.

The goal is then to show that there exists a typing derivation τ'_{vb} of the type judgement $Q'_1, Q_2 \vdash \langle M_a, M_b \rangle [V/x] : A_a \otimes A_b$ which satisfies that

$$\begin{aligned} \llbracket \tau'_{vb} \mid Q'_1, Q_2 \vdash \langle M_a, M_b \rangle [V/x] : A_a \otimes A_b \rrbracket = \\ \llbracket Q'_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0} \overline{M} \llbracket Q'_1 \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{f} \overline{M} F \llbracket A_a \otimes A_b \rrbracket. \end{aligned}$$

To prove that, we analyze the following three cases:

- $(x : B) \in !\Delta$, which means that $B = !B'$, $!\Delta = (x : !B')$, and $Q'_1 = Q_a, Q_b$:

Note that $Q_2 = \emptyset$ is deducible from $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket$ by Lemma 3.2.4 since $B = !B'$ is non-linear.

1. Induction hypotheses: there are typing derivations τ_{vb}^a for the type judgement $Q_a \vdash M_a[V/x] : A_a$ and τ_{vb}^b for $Q_b \vdash M_b[V/x] : A_b$ such that

$$\begin{aligned} h_a &= \llbracket \tau_{vb}^a \mid Q_a \vdash M_a[V/x] : A_a \rrbracket \\ &= \llbracket Q_a \rrbracket \xrightarrow{\text{id} \otimes g^0} \overline{M} \llbracket Q_a \rrbracket \otimes ! \llbracket B' \rrbracket \xrightarrow{\sigma} \overline{M} ! \llbracket B' \rrbracket \otimes \llbracket Q_a \rrbracket \xrightarrow{h^1} \overline{M} F \llbracket A_a \rrbracket \\ h_b &= \llbracket \tau_{vb}^b \mid Q_b \vdash M_b[V/x] : A_b \rrbracket \\ &= \llbracket Q_b \rrbracket \xrightarrow{\text{id} \otimes g^0} \overline{M} \llbracket Q_b \rrbracket \otimes !\Delta B' \xrightarrow{\sigma} \overline{M} ! \llbracket B' \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^2} \overline{M} F \llbracket A_b \rrbracket \end{aligned}$$

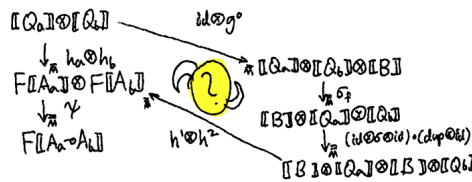
2. We have the following typing derivation:

$$\frac{Q_a \vdash M_a[V/x] : A_a \quad Q_b \vdash M_b[V/x] : A_b}{Q_a, Q_b \vdash \langle M_a, M_b \rangle [V/x] : A_a \otimes A_b} (\otimes_I)$$

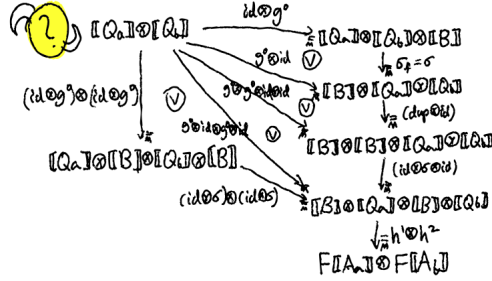
whose denotation is given by

$$\begin{aligned} \llbracket (\otimes_I); (\tau_{vb}^a, \tau_{vb}^b) \mid Q_a, Q_b \vdash \langle M_a, M_b \rangle [V/x] : A_a \otimes A_b \rrbracket = \\ \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h_a \otimes h_b} \overline{M} F \llbracket A_a \rrbracket \otimes F \llbracket A_b \rrbracket \xrightarrow{\psi} \overline{M} F (\llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket) \end{aligned}$$

Given these facts, we can derive the goal from following diagram.



where the commutativity in the middle is shown as follows:



- $(x : B) \in Q_a$, which means that $! \Delta = \emptyset$, $Q_a = Q'_a, (x : B)$, and $Q'_1 = Q'_a \otimes Q_b$:

In this case, $M_b[V/x] = M_b$ since $\mathbf{FV}(M_b) \subseteq \mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b)$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^a for the type judgement $Q'_a, Q_2 \vdash M_a[V/x] : A_a$ such that

$$\begin{aligned} h_a &= \llbracket \tau_{vb}^a \mid Q'_a, Q_2 \vdash M_a[V/x] : A_a \rrbracket \\ &= \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0} \overline{M} \llbracket Q'_a \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{h^1} \overline{M} F \llbracket A_a \rrbracket \end{aligned}$$

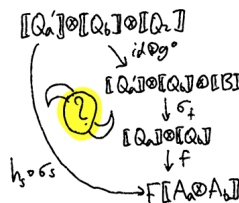
2. We have the following typing derivation:

$$\frac{Q'_a, Q_2 \vdash M_a[V/x] : A_a \quad Q_b \vdash M_b : A_b}{Q'_a, Q_2, Q_b \vdash \langle M_a, M_b \rangle[V/x] : A_a \otimes A_b} (\otimes_I)$$

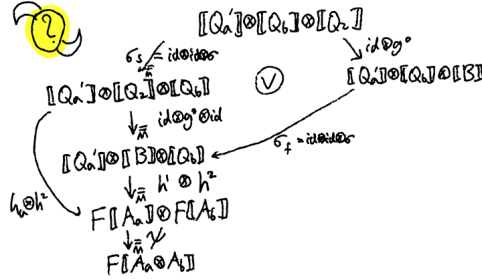
whose denotation is

$$\begin{aligned} h_s &= \llbracket (\otimes_I); (\tau_{vb}^a, \tau_2) \mid Q'_a, Q_2, Q_b \vdash \langle M_a, M_b \rangle[V/x] : A_a \otimes A_b \rrbracket \\ &= \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h_a \otimes h^2} \overline{M} F \llbracket A_a \rrbracket \otimes F \llbracket A_b \rrbracket \xrightarrow{\psi} \overline{M} F(\llbracket A_a \otimes A_b \rrbracket) \end{aligned}$$

From these facts, the goal can be derived from the following diagram.



where the commutativity in the center is proven in the following.



- $(x : B) \in Q_b$, which means that $! \Delta = \emptyset$, $Q_b = Q'_b, (x : B)$, and $Q'_1 = Q_a, Q'_b$:

In this case, $M_a[V/x] = M_a$ since $\mathbf{FV}(M_a) \subseteq \mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a)$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^b for the type judgement $Q'_b, Q_2 \vdash M_b[V/x] : A_b$ such that

$$\begin{aligned} & \llbracket \tau_{vb}^b \mid Q'_b, Q_2 \vdash M_b[V/x] : A_b \rrbracket \\ &= \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0} \overline{M} \llbracket Q'_b \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{h^2} \overline{M} F \llbracket A_b \rrbracket \end{aligned}$$

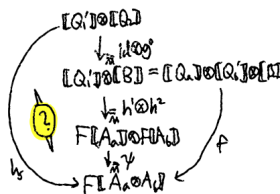
2. We have the following typing derivation:

$$\frac{Q_a \vdash M_a : A_a \quad Q'_b, Q_2 \vdash M_b[V/x] : A_b}{Q_a, Q'_b, Q_2 \vdash \langle M_a, M_b \rangle[V/x] : A_a \otimes A_b} (\otimes_I)$$

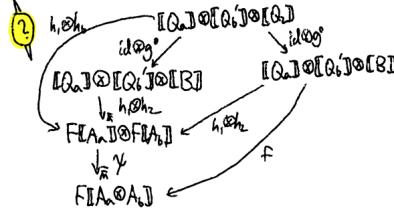
whose denotation is

$$\begin{aligned} h_s &= \llbracket (\otimes_I); (\tau_1, \tau_{vb}^b) \mid Q_a, Q'_b, Q_2 \vdash \langle M_a M_b \rangle[V/x] : A_a \otimes A_b \rrbracket \\ &= \llbracket Q_a \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{h^1 \otimes h_b} \overline{M} F \llbracket A_a \rrbracket \otimes F \llbracket A_b \rrbracket \xrightarrow{\psi} \overline{M} F \llbracket A_a \otimes A_b \rrbracket \end{aligned}$$

From these facts, the goal can be derived from the following diagram.



where the commutativity in the middle can be shown as follows.



$$\bullet \frac{!Δ, Q_a ⊢ M_a : A_a ⊗ A_b \quad !Δ, Q_b, (x' : A_a), (y : A_b) ⊢ M_b : A}{!Δ, Q_a, Q_b ⊢ \mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b : A} (\otimes_E):$$

Suppose that the typing derivation of $Q_1, (x : B) ⊢ M : A$ is obtained by applying (\otimes_E) -rule. Then, we know that $!Δ, Q_a, Q_b = Q_1, (x : B)$ and $M = \mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b$.

By definition, we have that

$$\begin{aligned} f' &= \llbracket (\otimes_E); (\tau_1, \tau_2) \mid !Δ, Q_a, Q_b ⊢ \mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b : A \rrbracket \\ &= \llbracket !Δ \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{\text{dup} \otimes \text{id}}_{\overline{M}} \llbracket !Δ \rrbracket \otimes \llbracket !Δ \rrbracket \otimes \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \\ &\quad \xrightarrow{\text{id} \otimes \sigma \otimes \text{id}}_{\overline{M}} \llbracket !Δ \rrbracket \otimes \llbracket Q_a \rrbracket \llbracket !Δ \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^1 \otimes \text{id}}_{\overline{M}} \\ &\quad F \llbracket A_a \otimes A_b \rrbracket \otimes \llbracket !Δ \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{t}_{\overline{M}} F(\llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket \otimes \llbracket !Δ \rrbracket \otimes \llbracket Q_b \rrbracket) \\ &\quad \xrightarrow{F(\sigma)}_{\overline{M}} F(\llbracket !Δ \rrbracket \otimes \llbracket Q_b \rrbracket \otimes \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket) \xrightarrow{F(h^2)}_{\overline{M}} F^2 \llbracket A \rrbracket \xrightarrow{\mu}_{\overline{M}} F \llbracket A \rrbracket \end{aligned}$$

where $h^1 = \llbracket \tau_1 \mid !Δ, Q_a ⊢ M_a : A_a \otimes A_b \rrbracket$ and $h^2 = \llbracket \tau_2 \mid !Δ, Q_b, (x' : A_a), (y : A_b) ⊢ M_b : A \rrbracket$.

For convenience, we define the morphism $f = f' \circ \sigma_f : \llbracket Q'_a \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket B \rrbracket \rightarrow_{\overline{M}} F \llbracket A \rrbracket$ which changes the order of the variables in the context.

By Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid Q_2 ⊢ V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket B \rrbracket$.

The goal is then to show that there exists a typing derivation τ'_{vb} of the type judgement $Q'_1, Q_2 ⊢ (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A$ which satisfies that

$$\begin{aligned} &\llbracket \tau'_{vb} \mid Q'_1, Q_2 ⊢ (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A \rrbracket \\ &= \llbracket Q'_1 \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket Q'_1 \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{f}_{\overline{M}} F \llbracket A \rrbracket. \end{aligned}$$

To prove that, we analyze the following three cases:

- $(x : B) \in !Δ$, which means that $B = !B'$, $!Δ = (x : !B')$, and $Q'_1 = Q_a, Q_b$:

Note that $Q_2 = \emptyset$ is deducible from $g = \llbracket \tau_g \mid Q_2 ⊢ V : B \rrbracket$ by Lemma 3.2.4 since $B = !B'$ is non-linear.

1. Induction hypotheses: there are typing derivations τ_{vb}^a for the type judgement $Q_a \vdash M_a[V/x] : A_a \otimes A_b$ and τ_{vb}^b for $Q_b, (x' : A_a), (y : A_b) \vdash M_b[V/x] : A$ such that

$$\begin{aligned}
 h_a &= [\tau_{vb}^a \mid Q_a \vdash M_a[V/x] : A_a \otimes A_b] \\
 &= [Q_a] \xrightarrow{\text{id} \otimes g^0} \overline{M} [Q_a] \otimes ! [B'] \xrightarrow{\sigma} \overline{M} ! [B'] \otimes [Q_a] \xrightarrow{h^1} \overline{M} F [A_a \otimes A_b] \\
 h_b &= [\tau_{vb}^b \mid Q_b, (x' : A_a), (y : A_b) \vdash M_b[V/x] : A] \\
 &= [Q_b] \otimes [A_a] \otimes [A_b] \xrightarrow{\text{id} \otimes g^0} \overline{M} [Q_b] \otimes [A_a] \otimes [A_b] \otimes ! \Delta B' \\
 &\quad \xrightarrow{\sigma} \overline{M} ! [B'] \otimes [Q_b] \otimes [A_a] \otimes [A_b] \xrightarrow{h^2} \overline{M} F [A]
 \end{aligned}$$

2. We know that

$$(\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] = (\mathbf{let} \langle x', y \rangle = M_a[V/x] \mathbf{in} M_b[V/x])$$

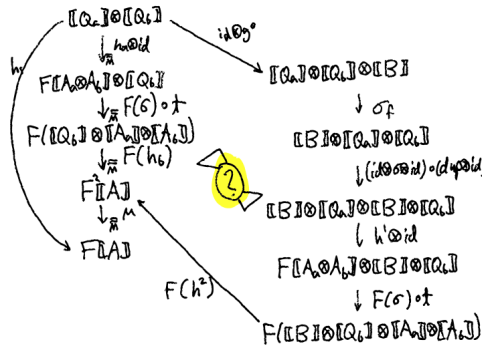
and we have the following typing derivation:

$$\frac{Q_a \vdash M_a[V/x] : A_a \otimes A_b \quad Q_b, (x' : A_a), (y : A_b) \vdash M_b[V/x] : A}{Q_a, Q_b \vdash (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A} (\otimes_E)$$

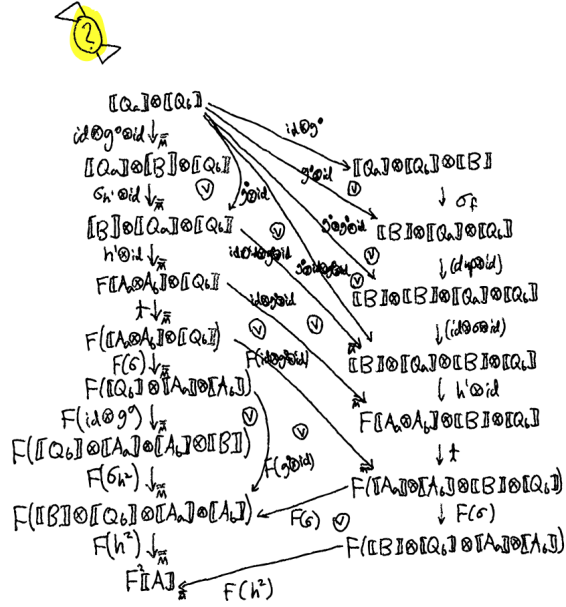
whose denotation is given by

$$\begin{aligned}
 h_s &= [(\otimes_E); (\tau_{vb}^a, \tau_{vb}^b) \mid Q_a, Q_b \vdash (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A] \\
 &= [Q_a] \otimes [Q_b] \xrightarrow{h_a \otimes \text{id}} \overline{M} F [A_a \otimes A_b] \otimes [Q_b] \\
 &\quad \xrightarrow{t} \overline{M} F ([A_a \otimes A_b] \otimes [Q_b]) \xrightarrow{F(\sigma)} \overline{M} F ([Q_b] \otimes [A_a] \otimes [A_b]) \\
 &\quad \xrightarrow{F(h_b)} \overline{M} F^2 [A] \xrightarrow{\mu} \overline{M} F [A]
 \end{aligned}$$

Given these facts, we can derive the goal from following diagram.



where the commutativity in the middle is shown as follows:



- $(x : B) \in Q_a$, which means that $! \Delta = \emptyset$, $Q_a = Q'_a, (x : B)$, and $Q'_1 = Q'_a \otimes Q_b$;
In this case, $M_b[V/x] = M_b$ since

$$\mathbf{FV}(M_b) \subseteq \mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b) \cup \{x', y\}$$

by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^a for the type judgement $Q'_a, Q_2 \vdash M_a[V/x] : A_a \otimes A_b$ such that

$$\begin{aligned} h_a &= \llbracket \tau_{vb}^a \mid Q'_a, Q_2 \vdash M_a[V/x] : A_a \otimes A_b \rrbracket \\ &= \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{\text{id} \otimes g^0} \llbracket Q'_a \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{h^1} \llbracket A_a \otimes A_b \rrbracket \end{aligned}$$

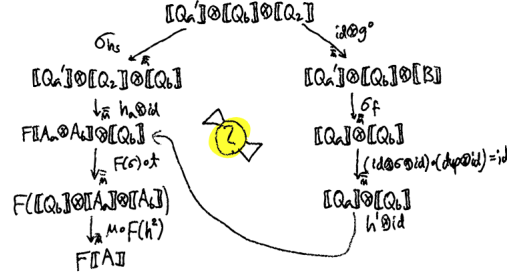
2. We have the following typing derivation:

$$\frac{Q'_a, Q_2 \vdash M_a[V/x] : A_a \otimes A_b \quad Q_b, (x' : A_a), (y : A_b) \vdash M_b : A}{Q'_a, Q_2, Q_b \vdash (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A} (\otimes E)$$

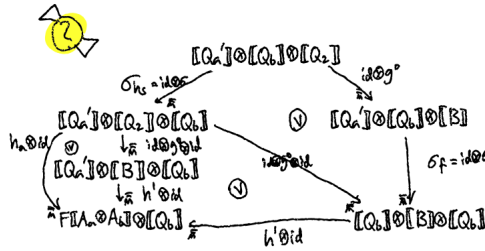
whose denotation is

$$\begin{aligned} h_s &= \llbracket (\otimes E); (\tau_{vb}^a, \tau_2) \mid Q'_a, Q_2, Q_b \vdash (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A \rrbracket \\ &= \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h_a \otimes \text{id}} \llbracket A_a \otimes A_b \rrbracket \otimes \llbracket Q_b \rrbracket \\ &\xrightarrow{t} \llbracket F(\llbracket A_a \otimes A_b \rrbracket) \otimes \llbracket Q_b \rrbracket \rrbracket \xrightarrow{F(\sigma)} \llbracket F(\llbracket Q_b \rrbracket) \otimes \llbracket A_a \rrbracket \otimes \llbracket A_b \rrbracket \rrbracket \\ &\xrightarrow{F(h^2)} \llbracket F^2 \llbracket A \rrbracket \rrbracket \xrightarrow{\mu} \llbracket F \llbracket A \rrbracket \rrbracket \end{aligned}$$

From these facts, the goal can be derived from the following diagram.



where the commutativity in the center is proven in the following.



- $(x : B) \in Q_b$, which means that $!\Delta = \emptyset$, $Q_b = Q'_b, (x : B)$, and $Q'_1 = Q_a, Q'_b$:

In this case, $M_a[V/x] = M_a$ since $\mathbf{FV}(M_a) \subseteq \mathbf{FV}(!\Delta) \cup \mathbf{FV}(Q_a)$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^b for the type judgement $Q'_b, (x' : A_a), (y : A_b), Q_2 \vdash M_b[V/x] : A$ such that

$$\begin{aligned} h_b &= \left[\tau_{vb}^b \mid Q'_b, (x' : A_a), (y : A_b), Q_2 \vdash M_b[V/x] : A \right] \\ &= \left[[Q'_b] \otimes [A_a] \otimes [A_b] \otimes [Q_2] \xrightarrow{\text{id} \otimes g^0} \overline{M} \right. \\ &\quad \left. [Q'_b] \otimes [A_a] \otimes [A_b] \otimes [B] \xrightarrow{h^2} \overline{M} F[A] \right] \end{aligned}$$

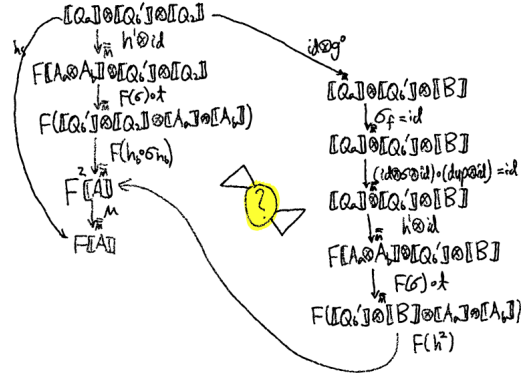
2. We have the following typing derivation:

$$\frac{Q_a \vdash M_a : A_a \otimes A_b \quad Q'_b, Q_2, (x' : A_a), (y : A_b) \vdash M_b[V/x] : A}{Q_a, Q'_b, Q_2 \vdash (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A} \quad (\otimes_E)$$

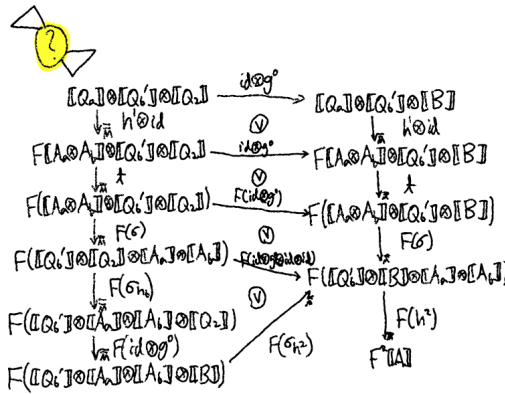
whose denotation is

$$\begin{aligned} h_s &= \left[(\otimes_E); (\tau_1, \tau_{vb}^b) \mid Q_a, Q'_b, Q_2 \vdash (\mathbf{let} \langle x', y \rangle = M_a \mathbf{in} M_b)[V/x] : A \right] \\ &= \left[[Q_a] \otimes [Q'_b] \otimes [Q_2] \xrightarrow{h^1 \otimes \text{id}} \overline{M} F[A_a \otimes A_b] \otimes [Q'_b] \otimes [Q_2] \right. \\ &\quad \xrightarrow{t} \overline{M} F([A_a \otimes A_b] \otimes [Q'_b] \otimes [Q_2]) \xrightarrow{F(\sigma)} \overline{M} F([Q'_b] \otimes [Q_2] \otimes [A_a] \otimes [A_b]) \\ &\quad \left. \xrightarrow{F(\sigma_{h_b})} \overline{M} F([Q'_b] \otimes [A_a] \otimes [A_b] \otimes [Q_2]) \xrightarrow{F(h_b)} \overline{M} F^2[A] \xrightarrow{\mu} \overline{M} F[A] \right] \end{aligned}$$

From these facts, the goal can be derived from the following diagram.



where the commutativity in the middle can be shown as follows.



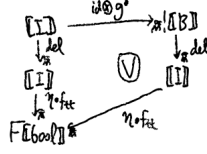
- $\frac{}{!\Delta \vdash \text{tt} : \text{bool}}(\text{tt})$:

In this case, the typing derivation τ_f of the type judgement $Q_1, (x : B) \vdash M : A$ is obtained by applying (tt)-rule. We let $!\Delta = Q_1, (x : B)$, which implies that $Q_1 = \emptyset$ and $B = !B'$ for some B' .

Then, by definition, $\llbracket (\text{tt}) \mid (x : !B') \vdash \text{tt} : \text{bool} \rrbracket = \llbracket !\Delta \rrbracket \xrightarrow{\text{del}}_{\overline{M}} \llbracket I \rrbracket \xrightarrow{\eta \circ f_{\text{tt}}} \llbracket B \rrbracket \xrightarrow{g^0} \llbracket !B' \rrbracket \xrightarrow{\eta} F \llbracket !B' \rrbracket$. Moreover, by Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid !\Delta', Q_2 \vdash V : B \rrbracket = \llbracket !\Delta' \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{g^0} \llbracket !B \rrbracket \xrightarrow{\eta} F \llbracket !B' \rrbracket$. Also, by Lemma 3.2.4, we know that $Q_2 = \emptyset$ since $B = !B'$ is non-linear. In addition, since B is non-linear, by Lemma 5.3.2, we know that $g^0 = f^{(n, g_0^0)}$ for some n and morphism g_0^0 , which implies that $g^0 = p(\dots)$.

Next, we know that $\text{tt}[V/x] = \text{tt}$ and, therefore, we can obtain for the typing derivation $\tau_{vb} = (\text{tt})$, the interpretation $\llbracket \tau_{vb} \mid \vdash \text{tt} : \text{bool} \rrbracket = \llbracket !\Delta' \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{\text{del}}_{\overline{M}} \llbracket I \rrbracket \xrightarrow{\eta \circ f_{\text{tt}}} \llbracket F \llbracket \text{bool} \rrbracket \rrbracket$. Then, it suffices to show the following

diagram commutes.



where the commute diagram in the middle follows from Lemma 4.5.4 since $[[!Δ]] = I$ is a parameter object and the morphisms dup and g^0 are morphisms of parameter objects of the form $p(\dots)$.

- $\frac{}{!Δ \vdash \text{ff} : \text{bool}}$ (ff):

Similar to the (tt) case.

- $\frac{!Δ, Q_a \vdash M : \text{bool} \quad !Δ, Q_b \vdash M_a : A \quad !Δ, Q_b \vdash M_b : A}{!Δ, Q_a, Q_b \vdash \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b : A}$ (if):

Suppose that the typing derivation of $Q_1, (x : B) \vdash M : A$ is obtained by applying (if)-rule. Then, we know that $!Δ, Q_a, Q_b = Q_1, (x : B)$ and $M = \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b$.

By definition, we have that

$$\begin{aligned}
 f' &= [(if); (\tau_0, \tau_1, \tau_2) \mid !Δ, Q_a, Q_b \vdash \mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b : A] \\
 &= [[!Δ]] \otimes [[Q_a]] \otimes [[Q_b]] \xrightarrow{\text{dup} \otimes \text{id}}_{\overline{M}} [[!Δ]] \otimes [[!Δ]] \otimes [[Q_a]] \otimes [[Q_b]] \\
 &\quad \xrightarrow{\text{id} \otimes \sigma \otimes \text{id}}_{\overline{M}} [[!Δ]] \otimes [[Q_a]] \otimes [[!Δ]] \otimes [[Q_b]] \xrightarrow{h \otimes \text{id}}_{\overline{M}} F[[\text{bool}]] \otimes [[!Δ]] \otimes [[Q_b]] \\
 &\quad \xrightarrow{t}_{\overline{M}} F([[!Δ]] \otimes [[Q_b]]) \xrightarrow{F(\star)}_{\overline{M}} F([[!Δ]] \otimes [[Q_b]]) + F([[!Δ]] + [[Q_b]]) \\
 &\quad \xrightarrow{F(h^1, h^2)}_{\overline{M}} F^2[[A]] \xrightarrow{\mu}_{\overline{M}} F[[A]]
 \end{aligned}$$

where $h = [\tau_0 \mid !Δ, Q_a \vdash M : \text{bool}]$, $h^1 = [\tau_1 \mid !Δ, Q_a \vdash M_a : A]$, and $h^2 = [\tau_2 \mid !Δ, Q_a \vdash M_b : A]$.

For convenience, we define the morphism $f = f' \circ \sigma_f : [[Q'_a]] \otimes [[Q'_b]] \otimes [[B]] \rightarrow_{\overline{M}} F[[A]]$ which changes the order of the variables in the context.

By Lemma 5.3.1, we have that $g = [\tau_g \mid Q_2 \vdash V : B] = [[Q_2]] \xrightarrow{g^0}_{\overline{M}} [[B]] \xrightarrow{\eta}_{\overline{M}} F[[B]]$.

The goal is then to show that there exists a typing derivation τ'_{vb} of the type judgement $Q'_1, Q_2 \vdash (\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b)[V/x] : A$ which satisfies that

$$\begin{aligned}
 &[[\tau'_{vb} \mid Q'_1, Q_2 \vdash (\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b)[V/x] : A]] \\
 &= [[Q'_1]] \otimes [[Q_2]] \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} [[Q'_1]] \otimes [[B]] \xrightarrow{f}_{\overline{M}} F[[A]].
 \end{aligned}$$

To prove that, we analyze the following three cases:

- $(x : B) \in !\Delta$, which means that $B = !B'$, $!\Delta = (x : !B')$, and $Q'_1 = Q_a, Q_b$:

Note that $Q_2 = \emptyset$ is deducible from $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket$ by Lemma 3.2.4 since $B = !B'$ is non-linear.

1. Induction hypotheses: there are typing derivations τ_{vb}^0 for the type judgement $Q_a \vdash M[V/x] : \text{bool}$, τ_{vb}^a for $Q_b \vdash M_a[V/x] : A$, and τ_{vb}^b for $Q_b \vdash M_b[V/x] : A$ such that

$$\begin{aligned} h_m &= \llbracket \tau_{vb}^0 \mid Q_a \vdash M[V/x] : \text{bool} \rrbracket \\ &= \llbracket Q_a \rrbracket \xrightarrow{\text{id} \otimes g^0} \llbracket Q_a \rrbracket \otimes ! \llbracket B' \rrbracket \xrightarrow{\sigma} ! \llbracket B' \rrbracket \otimes \llbracket Q_a \rrbracket \xrightarrow{h} F \llbracket \text{bool} \rrbracket \\ h_a &= \llbracket \tau_{vb}^a \mid Q_b \vdash M_a[V/x] : A \rrbracket \\ &= \llbracket Q_b \rrbracket \xrightarrow{\text{id} \otimes g^0} \llbracket Q_b \rrbracket \otimes ! \llbracket B' \rrbracket \xrightarrow{\sigma} ! \llbracket B' \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^1} F \llbracket A \rrbracket \\ h_b &= \llbracket \tau_{vb}^b \mid Q_b \vdash M_b[V/x] : A \rrbracket \\ &= \llbracket Q_b \rrbracket \xrightarrow{\text{id} \otimes g^0} \llbracket Q_b \rrbracket \otimes ! \llbracket B' \rrbracket \xrightarrow{\sigma} ! \llbracket B' \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h^2} F \llbracket A \rrbracket \end{aligned}$$

2. We know that

$$(\text{if } M \text{ then } M_a \text{ else } M_b)[V/x] = (\text{if } M[V/x] \text{ then } M_a[V/x] \text{ else } M_b[V/x])$$

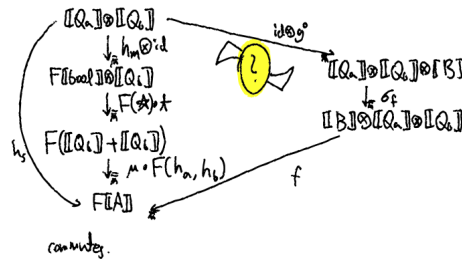
and we have the following typing derivation:

$$\frac{Q_a \vdash M[V/x] : \text{bool} \quad Q_b \vdash M_a[V/x] : A \quad Q_b \vdash M_b[V/x] : A}{Q_a, Q_b \vdash (\text{if } M \text{ then } M_a \text{ else } M_b)[V/x] : A} \text{ (if)}$$

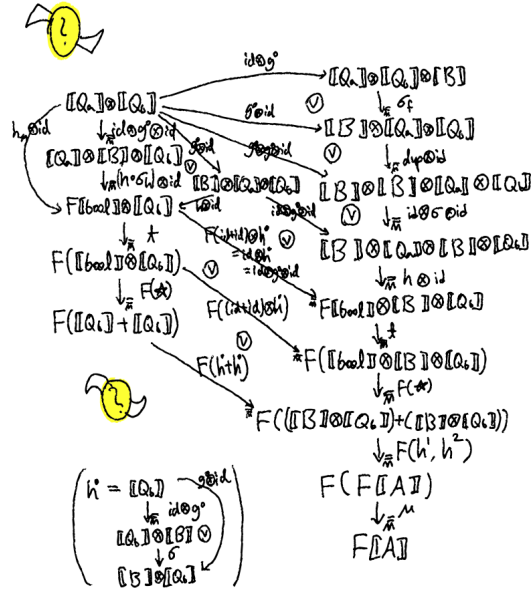
whose denotation is given by

$$\begin{aligned} h_s &= \llbracket (\text{if}); (\tau_{vb}^0, \tau_{vb}^a, \tau_{vb}^b) \mid Q_a, Q_b \vdash (\text{if } M \text{ then } M_a \text{ else } M_b)[V/x] : A \rrbracket \\ &= \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h_m \otimes \text{id}} F \llbracket \text{bool} \rrbracket \otimes \llbracket Q_b \rrbracket \\ &\xrightarrow{t} F(\llbracket \text{bool} \rrbracket \otimes \llbracket Q_b \rrbracket) \xrightarrow{F(\star)} F(\llbracket Q_b \rrbracket + \llbracket Q_b \rrbracket) \\ &\xrightarrow{F(h_a, h_b)} F^2 \llbracket A \rrbracket \xrightarrow{\mu} F \llbracket A \rrbracket \end{aligned}$$

Given these facts, we can derive the goal from following diagram.



where the commutativity in the middle is shown as follows:



Note that we obtain that $(h_a, h_b) = (h^1, h^1) \circ (h^0 + h^0)$, where $h^0 = g^0 \otimes id$, from Lemma 4.2.2.

- $(x : B) \in Q_a$, which means that $! \Delta = \emptyset$, $Q_a = Q'_a$, $(x : B)$, and $Q'_1 = Q'_a \otimes Q_b$:

In this case, we have $M_a[V/x] = M_a$ and $M_b[V/x] = M_b$ since $x \notin \mathbf{FV}(! \Delta, Q_b)$ and $\mathbf{FV}(M_a), \mathbf{FV}(M_b) \subseteq (\mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_b))$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^m for the type judgement $Q'_a, Q_2 \vdash M[V/x] : \text{bool}$ such that

$$\begin{aligned} h_m &= \llbracket \tau_{vb}^m \mid Q'_a, Q_2 \vdash M[V/x] : \text{bool} \rrbracket \\ &= \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{id \otimes g^0} \llbracket Q'_a \rrbracket \otimes \llbracket B \rrbracket \xrightarrow{h} F \llbracket \text{bool} \rrbracket \end{aligned}$$

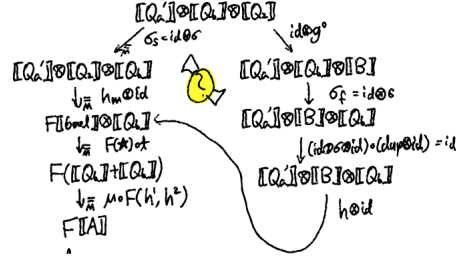
2. We have the following typing derivation:

$$\frac{Q'_a, Q_2 \vdash M[V/x] : \text{bool} \quad Q_b \vdash M_a : A \quad Q_b \vdash M_b : A}{Q'_a, Q_2, Q_b \vdash (\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b)[V/x] : A} \text{ (if)}$$

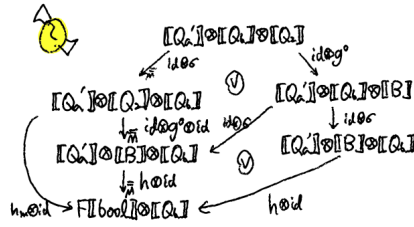
whose denotation is

$$\begin{aligned} h_s &= \llbracket (\mathbf{if}); (\tau_{vb}^m, \tau_1, \tau_2) \mid Q'_a, Q_2, Q_b \vdash (\mathbf{if} M \mathbf{then} M_a \mathbf{else} M_b)[V/x] : A \rrbracket \\ &= \llbracket Q'_a \rrbracket \otimes \llbracket Q_2 \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow{h_m \otimes id} \llbracket \text{bool} \rrbracket \otimes \llbracket Q_b \rrbracket \\ &\xrightarrow{t} F(\llbracket \text{bool} \rrbracket \otimes \llbracket Q_b \rrbracket) \xrightarrow{F(\star)} F(\llbracket Q_b \rrbracket + \llbracket Q_b \rrbracket) \\ &\xrightarrow{F(h^1, h^2)} F^2 \llbracket A \rrbracket \xrightarrow{\mu} F \llbracket A \rrbracket \end{aligned}$$

From these facts, the goal can be derived from the following diagram.



where the commutativity in the center is proven in the following.



- $(x : B) \in Q_b$, which means that $! \Delta = \emptyset$, $Q_b = Q'_b, (x : B)$, and $Q'_1 = Q_a, Q'_b$:

In this case, $M[V/x] = M$ since $\mathbf{FV}(M) \subseteq \mathbf{FV}(! \Delta) \cup \mathbf{FV}(Q_a)$ by Lemma 3.2.3.

1. Induction hypothesis: there exists a typing derivation τ_{vb}^a for the type judgement $Q'_b, Q_2 \vdash M_a[V/x] : A$ and τ_{vb}^b for $Q'_b, Q_2 \vdash M_b[V/x] : A$ such that

$$\begin{aligned}
 h_a &= \left[\tau_{vb}^a \mid Q'_b, Q_2 \vdash M_a[V/x] : A \right] \\
 &= \left[[Q'_b] \otimes [Q_2] \xrightarrow{\text{id} \otimes g^0} \overline{M} \left[[Q'_b] \otimes [B] \right] \xrightarrow{h^1} \overline{M} F[A] \right] \\
 h_b &= \left[\tau_{vb}^b \mid Q'_b, Q_2 \vdash M_b[V/x] : A \right] \\
 &= \left[[Q'_b] \otimes [Q_2] \xrightarrow{\text{id} \otimes g^0} \overline{M} \left[[Q'_b] \otimes [B] \right] \xrightarrow{h^2} \overline{M} F[A] \right]
 \end{aligned}$$

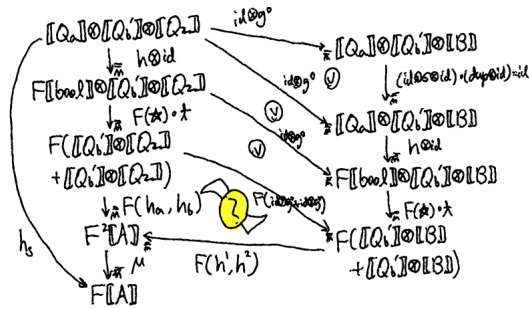
2. We have the following typing derivation:

$$\frac{Q_a \vdash M : \text{bool} \quad Q'_b, Q_2 \vdash M_a[V/x] : A \quad Q'_b, Q_2 \vdash M_b[V/x] : A}{Q_a, Q'_b, Q_2 \vdash (\text{if } M \text{ then } M_a \text{ else } M_b)[V/x] : A} \text{(if)}$$

whose denotation is

$$\begin{aligned}
h_s &= \left[\left[(\text{if}); (\tau_0, \tau_{ab}^a, \tau_{ab}^b) \mid Q_a, Q'_b, Q_2 \vdash (\mathbf{if} \ M \ \mathbf{then} \ M_a \ \mathbf{else} \ M_b)[V/x] : A \right] \right] \\
&= \llbracket Q_a \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \xrightarrow{h \otimes \text{id}}_{\overline{M}} F \llbracket \text{bool} \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \\
&\xrightarrow{t}_{\overline{M}} F(\llbracket \text{bool} \rrbracket \otimes \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket) \xrightarrow{F(\star)}_{\overline{M}} F(\llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket + \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket) \\
&\xrightarrow{F(h_a, h_b)}_{\overline{M}} F^2 \llbracket A \rrbracket \xrightarrow{\mu}_{\overline{M}} F \llbracket A \rrbracket
\end{aligned}$$

From these facts, the goal can be derived from the following diagram.



where the commutativity in the middle follows from the fact that $(h_a, h_b) = (h^1, h^2) \circ (\text{id} \otimes g^0 + \text{id} \otimes g^0)$ derived from Lemma 4.2.2 as follows.

$$\begin{aligned}
(h_a, h_b) &= \left(\llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket + \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \right) \rightarrow \llbracket A \rrbracket \\
&= \begin{pmatrix} \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket & \llbracket Q'_b \rrbracket \otimes \llbracket Q_2 \rrbracket \\ \downarrow \text{id} \otimes g^0 & \downarrow \text{id} \otimes g^0 \\ \llbracket Q'_b \rrbracket \otimes \llbracket B \rrbracket & \llbracket Q'_b \rrbracket \otimes \llbracket B \rrbracket \\ \downarrow h^1 & \downarrow h^2 \\ F \llbracket A \rrbracket & F \llbracket A \rrbracket \end{pmatrix} \\
&= (h^1, h^2) \circ (\text{id} \otimes g^0 + \text{id} \otimes g^0)
\end{aligned}$$

- $\overline{! \Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A)}$ (box):

In this case, we assume that the typing derivation τ_f is (box). It follows that $Q_1 = \emptyset$ and $B = !B'$, for some B' , from $! \Delta = Q_1, (x : B)$.

By definition, we have that

$$\begin{aligned}
f &= \llbracket (\text{box}) \mid ! \Delta \vdash \text{box}_P : !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket \\
&= \llbracket ! \Delta \rrbracket \xrightarrow{\text{del}}_{\overline{M}} I \xrightarrow{\eta_{! [P \multimap A]}(I)}_{\overline{M}} ! [P \multimap A] \multimap (I \otimes ! [P \multimap A]) \\
&\xrightarrow{(! [P \multimap A] \multimap -)(\eta \circ \text{box})}_{\overline{M}} ! [P \multimap A] \multimap F(! \llbracket \text{QChan}(P, A) \rrbracket) \\
&\xrightarrow{\eta}_{\overline{M}} F \llbracket !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket.
\end{aligned}$$

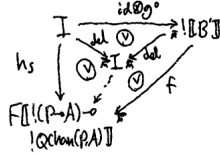
By Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{\overline{g^0}} \overline{M} \llbracket B \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket B \rrbracket$. Note that $Q_2 = \emptyset$ by Lemma 3.2.4 since $B = !B'$ is non-linear. Also, since B is non-linear, by Lemma 5.3.2, we know that $g^0 = f^{(n, g_0^0)}$ for some n and morphism g_0^0 , which implies that $g^0 = p(\dots)$. Next, since $\text{box}_P[V/x] = \text{box}_P$, we have the following derivation:

$$\overline{\vdash \text{box}_P[V/x] : !(P \multimap A) \multimap !\text{QChan}(P, A)} \text{ (box)}$$

whose denotation is

$$\begin{aligned} h_s &= \llbracket (\text{box}) \mid \vdash \text{box}_P[V/x] : !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket \\ &= I \xrightarrow{\text{del}} \overline{M} I \xrightarrow{\eta_{\llbracket P \multimap A \rrbracket}^{(I)}} \overline{M} ! \llbracket P \multimap A \rrbracket \multimap (I \otimes ! \llbracket P \multimap A \rrbracket) \\ &\quad \xrightarrow{(! \llbracket P \multimap A \rrbracket \multimap \multimap)(\eta \circ \text{box})} \overline{M} ! \llbracket P \multimap A \rrbracket \multimap F(! \llbracket \text{QChan}(P, A) \rrbracket)} \\ &\xrightarrow{\eta} \overline{M} F \llbracket !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket \end{aligned}$$

Then, we can show the goal as follows.



where the dots represent the morphism

$$\begin{aligned} I &\xrightarrow{\eta_{\llbracket P \multimap A \rrbracket}^{(I)}} \overline{M} ! \llbracket P \multimap A \rrbracket \multimap (I \otimes ! \llbracket P \multimap A \rrbracket) \\ &\quad \xrightarrow{(! \llbracket P \multimap A \rrbracket \multimap \multimap)(\eta \circ \text{box})} \overline{M} ! \llbracket P \multimap A \rrbracket \multimap F(! \llbracket \text{QChan}(P, A) \rrbracket)} \\ &\xrightarrow{\eta} \overline{M} F \llbracket !(P \multimap A) \multimap !\text{QChan}(P, A) \rrbracket \end{aligned}$$

- $\overline{! \Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A)}$ (unbox):

In this case, we assume that the typing derivation τ_f is (unbox). Note that $Q_2 = \emptyset$ by Lemma 3.2.4 since $B = !B'$ is non-linear. It follows that $Q_1 = \emptyset$ and $B = !B'$, for some B' , from $! \Delta = Q_1, (x : B)$.

By definition, we have that

$$\begin{aligned} f &= \llbracket (\text{unbox}) \mid ! \Delta \vdash \text{unbox} : \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket \\ &= \llbracket ! \Delta \rrbracket \xrightarrow{\text{del}} \overline{M} I \xrightarrow{\eta_{\llbracket \text{QChan}(P, A) \rrbracket}^{(I)}} \overline{M} \llbracket \text{QChan}(P, A) \rrbracket \multimap (I \otimes \llbracket \text{QChan}(P, A) \rrbracket)} \\ &\quad \xrightarrow{(\llbracket \text{QChan}(P, A) \rrbracket \multimap \multimap)(\eta \circ \text{unbox})} \overline{M} \llbracket \text{QChan}(P, A) \rrbracket \multimap F \llbracket P \multimap A \rrbracket} \\ &\xrightarrow{\eta} \overline{M} F \llbracket \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket. \end{aligned}$$

By Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{\overline{g^0}}_{\overline{M}}$
 $\llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket B \rrbracket$. Also, since B is non-linear, by Lemma 5.3.2, we know
that $g^0 = f^{(n, g_0^0)}$ for some n and morphism g_0^0 , which implies that $g^0 =$
 $p(\dots)$.

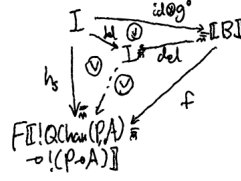
Next, since $\text{unbox}[V/x] = \text{unbox}$, we have the following derivation:

$$\overline{\vdash \text{unbox}[V/x] : \text{QChan}(P, A) \multimap (P \multimap A)} \text{ (unbox)}$$

whose denotation is

$$\begin{aligned} h_s &= \llbracket (\text{unbox}) \mid \vdash \text{unbox}[V/x] : \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket \\ &= I \xrightarrow{\text{del}}_{\overline{M}} I \xrightarrow{\eta_{\llbracket \text{QChan}(P, A) \rrbracket}^{(I)}}_{\overline{M}} \llbracket \text{QChan}(P, A) \rrbracket \multimap (I \otimes \llbracket \text{QChan}(P, A) \rrbracket) \\ &\quad \xrightarrow{(\llbracket \text{QChan}(P, A) \rrbracket \multimap -)(\eta \circ \text{unbox})}_{\overline{M}} \llbracket \text{QChan}(P, A) \rrbracket \multimap F \llbracket P \multimap A \rrbracket \\ &\xrightarrow{\eta}_{\overline{M}} F \llbracket \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket \end{aligned}$$

Then, we can show the goal as follows.



where the dots represent the morphism

$$\begin{aligned} I &\xrightarrow{\eta_{\llbracket \text{QChan}(P, A) \rrbracket}^{(I)}}_{\overline{M}} \llbracket \text{QChan}(P, A) \rrbracket \multimap (I \otimes \llbracket \text{QChan}(P, A) \rrbracket) \\ &\quad \xrightarrow{(\llbracket \text{QChan}(P, A) \rrbracket \multimap -)(\eta \circ \text{unbox})}_{\overline{M}} \llbracket \text{QChan}(P, A) \rrbracket \multimap F \llbracket P \multimap A \rrbracket \\ &\xrightarrow{\eta}_{\overline{M}} F \llbracket \text{QChan}(P, A) \multimap (P \multimap A) \rrbracket \end{aligned}$$

- $\frac{\gamma_a \vdash m_a : A \quad \gamma_b \vdash m_b : A}{\gamma_a \times \gamma_b \vdash [m_a, m_b] : A} \text{ (b):}$

This case does not match the type judgement $! \Delta, Q_2, (x : B) \vdash M : A$.

- $\frac{p \vDash P \quad \mathbf{vBind}(! \Delta, \mathbf{out}(Q), m, A)}{! \Delta \vdash (p, Q, m) : ! \text{QChan}(P, A)} \text{ (QChan}_I \text{):}$

Suppose that the typing derivation of $Q_1, (x : B) \vdash M : A$ is obtained by applying (QChan_I)-rule. Then, we know that $! \Delta = Q_1, (x : B)$ implies that $Q_1 = \emptyset$ and $B = !B'$ for some B' .

By definition, we have that

$$\begin{aligned}
f &= \llbracket (\text{QChan}_I); (\tau_i)_{i:\text{leaf}} \mid (x : !B') \vdash (p, Q, m) : !\text{QChan}(P, A) \rrbracket \\
&= \llbracket !B' \rrbracket \xrightarrow{\delta}_{\overline{M}} !! \llbracket B' \rrbracket \xrightarrow{!(\eta_{\llbracket P \rrbracket} \llbracket !B' \rrbracket})}_{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} (\llbracket !B' \rrbracket \otimes \llbracket P \rrbracket)) \\
&= \frac{!(\llbracket P \rrbracket \multimap -)(\text{id} \otimes q)}{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} (\llbracket !B' \rrbracket \otimes F \llbracket \text{out}(Q) \rrbracket)) \\
&= \frac{!(\llbracket P \rrbracket \multimap -)(t')}{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} F(\llbracket !B' \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket)) \\
&= \frac{!(\llbracket P \rrbracket \multimap -)(\mu \circ F(h))}{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} F \llbracket A \rrbracket) \xrightarrow{\eta^{\text{obox}}}_{\overline{M}} F(! \llbracket \text{QChan}(P, A) \rrbracket)
\end{aligned}$$

where

$$h = \llbracket (\tau_i)_{i:\text{leaf}} \mid \mathbf{vBind}(!B', \text{out}(Q), m, A) \rrbracket$$

and

$$q = \llbracket p \rrbracket \rightarrow_{\overline{M}} \llbracket \text{in}(Q) \rrbracket \xrightarrow{\llbracket Q \rrbracket}_{\overline{M}} F \llbracket \text{out}(Q) \rrbracket.$$

By Lemma 5.3.1, we have that $g = \llbracket \tau_g \mid Q_2 \vdash V : B \rrbracket = \llbracket Q_2 \rrbracket \xrightarrow{g^0}_{\overline{M}} \llbracket B \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket B \rrbracket$. Note that $Q_2 = \emptyset$ by Lemma 3.2.4 since $B = !B'$ is non-linear. Also, since B is non-linear, by Lemma 5.3.2, we know that $g^0 = f^{(n, g_0^0)}$ for some n and morphism g_0^0 , which implies that $g^0 = p(\dots)$.

Moreover, we also know the followings:

1. induction hypothesis:

Since $(p, Q, m)[V/x] = (p, Q, m[V/x])$, $\mathbf{vBind}((x : !B'), \text{out}(Q), m, A)$, and $\vdash V : !B'$, by the induction hypothesis, we know that there exists a typing derivation τ_{vb} of $\mathbf{vBind}(\emptyset, \text{out}(Q), m[V/x], A)$ such that

$$\begin{aligned}
h' &= \llbracket \tau_{vb} \mid \mathbf{vBind}(\emptyset, \text{out}(Q), m[V/x], A) \rrbracket \\
&= \llbracket \text{out}(Q) \rrbracket \otimes I \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket \text{out}(Q) \rrbracket \otimes \llbracket !B' \rrbracket \xrightarrow{\sigma}_{\overline{M}} \\
&\quad \llbracket !B' \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket \xrightarrow{h}_{\overline{M}} F \llbracket A \rrbracket
\end{aligned}$$

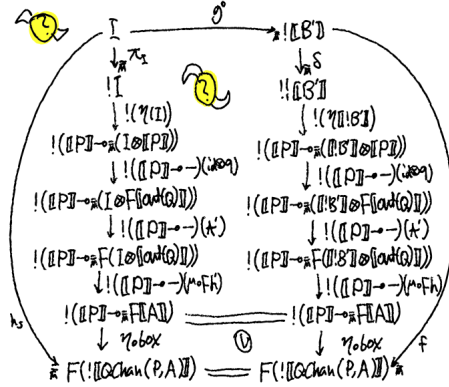
2. typing derivation:

$$\frac{p \vdash P \quad \mathbf{vBind}(\emptyset, \text{out}(Q), m[V/x], A)}{!\Delta \vdash (p, Q, m)[V/x] : !\text{QChan}(P, A)} (\text{QChan}_I)$$

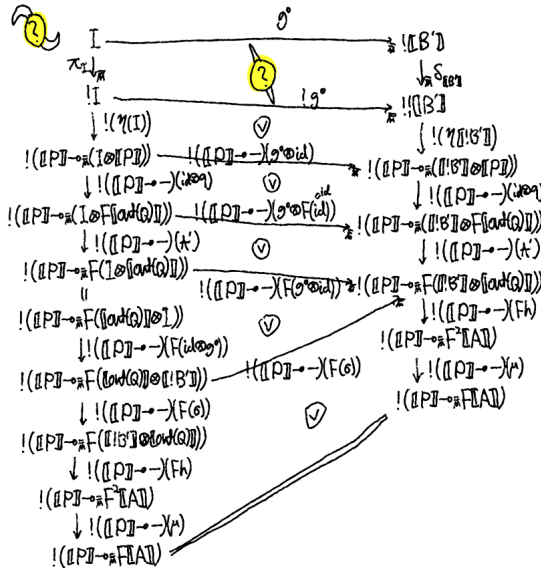
whose denotation is

$$\begin{aligned}
h_s &= \llbracket (\text{QChan}_I); (\tau_{vb}) \mid (p, Q, m)[V/x] : !\text{QChan}(P, A) \rrbracket \\
&= I \xrightarrow{\pi_I}_{\overline{M}} !I \xrightarrow{!(\eta_{\llbracket P \rrbracket}(I))}_{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} (I \otimes \llbracket P \rrbracket)) \\
&= \frac{!(\llbracket P \rrbracket \multimap -)(\text{id} \otimes q)}{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} (I \otimes F \llbracket \text{out}(Q) \rrbracket)) \\
&= \frac{!(\llbracket P \rrbracket \multimap -)(t')}{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} F(I \otimes \llbracket \text{out}(Q) \rrbracket)) \\
&= \frac{!(\llbracket P \rrbracket \multimap -)(\mu \circ F(h'))}{\overline{M}} !(\llbracket P \rrbracket \multimap_{\overline{M}} F \llbracket A \rrbracket) \xrightarrow{\eta^{\text{obox}}}_{\overline{M}} F(! \llbracket \text{QChan}(P, A) \rrbracket)
\end{aligned}$$

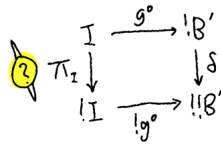
The goal is then to show that the denotation h_s of the type derivation $(QChan_I); (\tau_{vb})$ is equivalent to $I \xrightarrow{g^0} \overline{M} \llbracket B' \rrbracket \xrightarrow{f} \overline{M}$. It can be shown as follows.



where the commutativity in the middle can be shown as follows.



Note that the following commutativity follows from Lemma 5.3.3.



vBind part Proof by induction on the type derivation τ_g of the judgement $\mathbf{vBind}((x : !A'), c, m[V/x], A)$

$$\frac{Q \cap \mathbf{FV}(!\Delta) = \emptyset \quad !\Delta, \mathbf{TC}_Q(Q) \vdash M : A}{\mathbf{vBind}(!\Delta, Q, M, A)} (\mathbf{vBind}_{nb}):$$

In this case, we know that $! \Delta = (x : !A')$, $c = Q$, $m = M$, and that

$$\begin{aligned} h &= \llbracket \tau_g \mid \mathbf{vBind}((x : !A'), c, m[V/x], A) \rrbracket \\ &= \llbracket \tau_g \mid (x : !A'), \mathbf{TC}_Q(Q) \vdash M : A \rrbracket. \end{aligned}$$

By Lemma 5.3.1, we have that $g = \llbracket \tau_v \mid \vdash V : !A' \rrbracket = I \xrightarrow{g^0} \overline{M} \llbracket !A' \rrbracket \xrightarrow{\eta} \overline{M} F \llbracket !A' \rrbracket$.

Then, we have the following:

1. Induction hypothesis:

By the typing derivation part of the substitution lemma, we know that there exists a typing derivation τ_{vb} of the type judgement $\mathbf{TC}_Q(Q) \vdash M[V/x] : A$ such that

$$\begin{aligned} h_a &= \llbracket \tau_{vb} \mid \mathbf{TC}_Q(Q) \vdash M[V/x] : A \rrbracket \\ &= \llbracket \mathbf{TC}_Q(Q) \rrbracket \xrightarrow{\text{id} \otimes g^0} \overline{M} \llbracket \mathbf{TC}_Q(Q) \rrbracket \otimes \llbracket !A' \rrbracket \xrightarrow{h} \overline{M} F \llbracket A \rrbracket. \end{aligned}$$

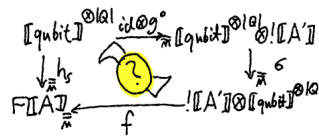
2. Derivation rule:

$$\frac{Q \cap \mathbf{FV}(\emptyset) = \emptyset \quad \mathbf{TC}_Q(Q) \vdash M[V/x] : A}{\mathbf{vBind}(\emptyset, Q, M[V/x], A)} (\mathbf{vBind}_{nb})$$

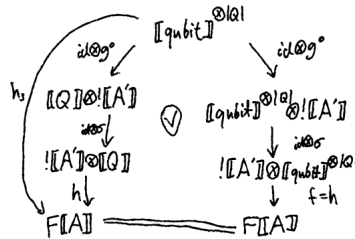
whose denotation is

$$h_s = \llbracket \tau_{vb} \mid \mathbf{vBind}(\emptyset, Q, M[V/x], A) \rrbracket = \llbracket \mathbf{qubit} \rrbracket^{\otimes |Q|} \xrightarrow{h_a} \overline{M} F \llbracket A \rrbracket$$

Finally, we show the following goal



which follows from the following.



$$\bullet \frac{\mathbf{vBind}(!\Delta, c_a, m_a, A) \quad \mathbf{vBind}(!\Delta, c_b, m_b, A)}{\mathbf{vBind}(!\Delta, [c_a, c_b], [m_a, m_b], A)} (\mathbf{vBind}_b):$$

In this case, we have that $!\Delta = (x : !A')$, $c = [c_a, c_b]$, and $m = [m_a, m_b]$.
 By definition, $\llbracket \tau_i + + \tau_j \mid \mathbf{vBind}((x : !A'), [c_a, c_b], [m_a, m_b], A) \rrbracket = \llbracket !A' \rrbracket \otimes (\llbracket c_a \rrbracket + \llbracket c_b \rrbracket) \xrightarrow{\star'}_{\overline{M}} \llbracket !A' \rrbracket \otimes \llbracket c_a \rrbracket + \llbracket !A' \rrbracket \otimes \llbracket c_b \rrbracket \xrightarrow{(h^1, h^2)}_{\overline{M}} F \llbracket A \rrbracket$, where $h^1 = \llbracket \tau_i \mid \mathbf{vBind}((x : !A'), c_a, m_a, A) \rrbracket$ and $h^2 = \llbracket \tau_j \mid \mathbf{vBind}((x : !A'), c_b, m_b, A) \rrbracket$.

By Lemma 5.3.1, we have that $g = \llbracket \tau_v \mid V : !A' \rrbracket = I \xrightarrow{g^0}_{\overline{M}} \llbracket !A' \rrbracket \xrightarrow{\eta}_{\overline{M}} F \llbracket !A' \rrbracket$.

Then, we have the following:

1. Induction hypothesis:

There exists a typing derivation τ_{vb}^i of the judgement

$\mathbf{vBind}(\emptyset, c_a, m_a[V/x], A)$ and τ_{vb}^j of the judgement

$\mathbf{vBind}(\emptyset, c_b, m_b[V/x], A)$ which satisfies the following equalities:

$$\begin{aligned} h_a &= \llbracket \tau_{vb}^i \mid \mathbf{vBind}(\emptyset, c_a, m_a[V/x], A) \rrbracket \\ &= \llbracket c_a \rrbracket \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket c_a \rrbracket \otimes \llbracket !A' \rrbracket \xrightarrow{\sigma}_{\overline{M}} \llbracket !A' \rrbracket \otimes \llbracket c_a \rrbracket \xrightarrow{h^1}_{\overline{M}} F \llbracket A \rrbracket \\ h_b &= \llbracket \tau_{vb}^j \mid \mathbf{vBind}(\emptyset, c_b, m_b[V/x], A) \rrbracket \\ &= \llbracket c_b \rrbracket \xrightarrow{\text{id} \otimes g^0}_{\overline{M}} \llbracket c_b \rrbracket \otimes \llbracket !A' \rrbracket \xrightarrow{\sigma}_{\overline{M}} \llbracket !A' \rrbracket \otimes \llbracket c_b \rrbracket \xrightarrow{h^2}_{\overline{M}} F \llbracket A \rrbracket \end{aligned}$$

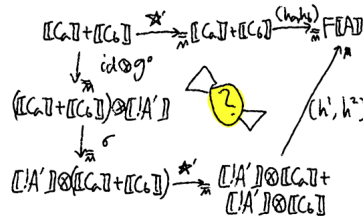
2. Derivation rule: since $[m_a, m_b][V/x] = [m_a[V/x], m_b[V/x]]$, from the following derivation rule

$$\frac{\mathbf{vBind}(\emptyset, c_a, m_a[V/x], A) \quad \mathbf{vBind}(\emptyset, c_b, m_b[V/x], A)}{\mathbf{vBind}(\emptyset, [c_a, c_b], [m_a, m_b][V/x], A)} (\mathbf{vBind}_b)$$

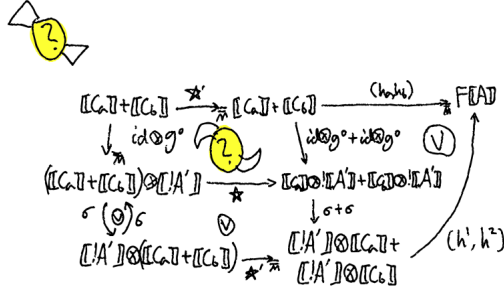
we can obtain its denotation.

$$\begin{aligned} &\llbracket \tau_{vb}^i + + \tau_{vb}^j \mid \mathbf{vBind}(\emptyset, [c_a, c_b], [m_a, m_b][V/x], A) \rrbracket \\ &= \llbracket c_a \rrbracket + \llbracket c_b \rrbracket \xrightarrow{\star'}_{\overline{M}} \llbracket c_a \rrbracket + \llbracket c_b \rrbracket \xrightarrow{(h_a, h_b)} F \llbracket A \rrbracket \end{aligned}$$

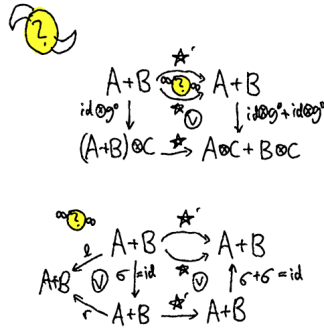
Finally, we show the following goal



which follows from the followings:



and



where $(h_a, h_b) = (h^1, h^2) \circ (\sigma + \sigma) \circ (\text{id} \otimes g^0 + \text{id} \otimes g^0)$ is derived from Lemma 4.2.2.

□

Finally, we show the preservation of categorical semantics over the reduction of typing derivations by the circuit-buffering operational semantics. In general, preservation of categorical semantics states that the categorical interpretation of the typing derivation is preserved over the reduction. However, in our type system, a term can have more than one type judgement and there can be multiple type derivations for each type judgement. Therefore, our preservation theorem states that for a type judgement and a typing derivation, there exists certain typing judgement of the reduced type judgement which has the same interpretation of the original typing derivation.

Theorem 5.3.5 (Preservation). *For any two configurations (Q_1, m_1) and (Q_2, m_2) such that $(Q_1, m_1) \rightarrow (Q_2, m_2)$, if $\vdash (Q_1, m_1) : A$, then for any typing derivation π_1 of $\vdash (Q_1, m_1) : A$, there exists a typing derivation π_2 of $\vdash (Q_2, m_2) : A$ such that $\llbracket \pi_1 \rrbracket = \llbracket \pi_2 \rrbracket$.*

Proof. Proof can be found in Appendix A.

□

In fact, the preservation theorem allows us to prove the soundness theorem of the categorical semantics when we only consider the typing derivations of basic types. The soundness theorem states that: for all typing derivations of two terms of the same basic type, if they reduce to the same value, then the denotations of the typing derivations are equal. The proof follows from the definition of basic type, the termination property of the operational semantics, and the preservation theorem.

Corollary 5.3.5.1 (Soundness). *For any typing derivation τ_1 and τ_2 of the respective type judgements $\vdash (Q_1, M_1) : A$ and $\vdash (Q_2, M_2) : A$ for a basic type A , if there is some (Q, V) such that $(Q_1, M_1) \rightarrow^* (Q, V)$ and $(Q_2, M_2) \rightarrow^* (Q, V)$, then $\llbracket \tau_1 \vdash (Q_1, M_1) : A \rrbracket = \llbracket \tau_2 \vdash (Q_2, M_2) : A \rrbracket$. \square*

6 - Conclusion and Discussion

In this thesis, we formalize the semantics of a programming language for quantum channel construction based on the QRAM model of quantum computation. We defined the algebraic structure of quantum computation (Definition 3.1.4) and compared it with the algebraic structure of quantum computation by Sam Staton in [70]. Then, we formally defined the language Proto-Quipper-L. The language is an extension of lambda-calculus: we include quantum channel constants and box and unbox operators, like in other circuit description languages. Then the language is again extended by branching terms to reflect the quantum channel's algebraic structure and formalize the operational semantics of quantum channel operators, i.e., box and unbox. In the language, quantum operators like qubit initialization, unitary maps, and measurements are defined as constant terms by the quantum channel operators applied to predefined quantum channel constants. Lastly, the branching term appears when the branching structure of the quantum channel, which is introduced by measurement, is lifted to the classical computation by unboxing.

Next, to avoid programs that violate the no-cloning theorem of quantum states, we defined a linear and non-linear type system capable of representing both linear and non-linear variables in the term. To accord with the structure of the branching term, typing context and type judgment are extended with the branching structure. However, the type system assures that each term in the leaf of the branching structure has the same type. Typing derivation is defined by following the intuitionistic multiplicative part of linear logic.

Furthermore, we defined two operational semantics: circuit-buffering and QRAM-based operational semantics. On the one hand, circuit-buffering semantics formalizes the interaction between the classical host and the quantum co-processor. The classical host buffers quantum operators to the quantum channel object and sends it to the quantum co-processor. The quantum co-processor creates a branching structure in the computation. In particular, the branching side-effect of measurement is formalized in two folds: first, by adding a measurement operator to the quantum object in the buffer, and second, by creating a branching structure in the term. On the other hand, QRAM-based operational semantics simulates the quantum operators on the quantum state and reduces the branching structure into a mixed state of the quantum state. Both of the operational semantics are shown to be type-safe—which means that the reduction of a well-typed term preserves the type (subject reduction lemma), and each well-typed term reduces to a value in finite steps (progress lemma and termination lemma).

Next, in Chapter 4, we presented the categorical semantics of Proto-Quipper-L. The categorical model is built based on Benton's linear and non-

linear category, Rios and Selinger’s coproduct completion, and Moggi’s computational model with the non-deterministic monad. The categorical semantics allows us to interpret each typing derivation of the Proto-Quipper-L program in terms of a morphism of the categorical model. Furthermore, we showed a concrete model of the language (i.e., the category of the diagrams), which means that the axioms induced by the aforementioned categorical model are satisfied by the concrete model. Consequently, we obtain an interpretation of the well-typed term of the language into a family of circuits indexed by set. We then showed that our categorical model reflects the meaning of the program given by the circuit-buffering operational semantics by showing that the interpretation of a well-typed term is preserved over the reduction and by proving the soundness theorem over the basic type as a corollary.

The argument that we used to prove the soundness theorem (restricted to basic types) relies on the notion of basic type and the preservation theorem. The soundness theorem states that if two configurations (of two terms in the same context) reduce to the same configuration of value, then the denotations of the configurations are equivalent. For simplicity, we weakened the theorem in this thesis by restricting the two configurations by the two terms in the empty context. Given that all reduction terminates, the preservation theorem implies that each denotation of each configuration of a value of basic type (note that there can be multiple denotations) represents a subset of configurations that reduce to the configuration of the same value—i.e., the subset of configuration whose denotation is equal the given denotation of the configuration of the value. Given these, proving the soundness theorem is equivalent to showing that all denotations of each configuration of value of basic type are the same. This was when we introduced the notion of basic type, which is precisely the type that satisfies the necessary condition to prove the soundness theorem.

Moreover, the uniqueness of each denotation of any configuration of basic type implies adequacy. Similar to the soundness, the adequacy (which is the converse of the soundness) corresponds to the proof that each denotation of the configuration of value is unique, meaning that the equivalence of the denotations of configurations implies the observational equivalence. Therefore, the uniqueness of the denotation of each configuration of the value of basic type implies adequacy. Hence, we can obtain the full-abstraction theorem for the configurations of the terms of basic type from the soundness and the adequacy theorems.

In addition, given that the interpretation in the semantics depends on the type derivation of well-typed terms, the possibility of different typing derivations of the same term arises the possibility of different denotations for the same configuration. Therefore, it would simplify the proofs when we distinguish the terms with different typing derivations by introducing new term

constructors for promotion and dereliction and marking type for each variable in the term. In this case, the term precisely represents the typing derivation, and the interpretation of a well-typed configuration is a function–meaning that there is one denotation for each configuration. In this case, it should be noticed that we may need to expand the observational equivalence to include the pair of terms that only differ by the consequent applications of a promotion and a dereliction.

Otherwise, for convenience, we may keep all well-typed values with new constructors as value and do not add any additional reduction rules to eliminate unnecessary promotions and derelictions. Then, the arguments on the properties of categorical semantics introduced above can be rewritten:

- The preservation theorem states that all configuration that reduces to the same configuration of value has the same denotation, which we call the set of configuration named by the configuration of value;
- The equivalence class of configuration over the observational equivalence is represented by the configurations of value;
- To show the soundness theorem, it suffices to show that the configurations of value with the same meaning (or the original value without new constructors) have the same denotation; and
- To show the adequacy theorem, we only need to prove that the denotations of the configurations of different values are not equivalent.

In the future, we would like to extend the expressivity of the language and the type system by introducing inductive types and dependent types in the type system. Categorical semantics of these features have already been studied in various works [38, 39, 24, 23]. These works provide abstract categorical structures required to introduce such features of programming language in the categorical semantics. Therefore, one approach to extending our programming language is to equip the categorical model with such a structure. The consistency of the model could be verified by finding a concrete category that satisfies the required conditions. One candidate would be the coproduct completion of the category of diagrams that we defined in this thesis.

Another candidate would be the category of presheaf over the category $\overline{\overline{M}}$, the categorical model of our language. The presheaf category, in this case, would be the category of functors from $\overline{\overline{M}}$ to the category **Set**. Remembering that a quantum state can be represented as a function from the self-adjoint operators to the real interval $[0, 1]$ and the fact the morphism in the categorical model $\overline{\overline{M}}$ of the language corresponds to the family of diagrams each of which represents a quantum operator. Therefore, we can say that the presheaf category **Set** $^{\overline{\overline{M}}}$ contains the quantum states.

Furthermore, Moggi explains in his paper [42] that we can lift the computational model (monad) over a category \mathcal{C} into a computational model over the presheaf over \mathcal{C} . Since the presheaf category is topos, it has an internal language (or dependent type system). Applying it to the category $\overline{\overline{M}}$, the presheaf category will generate a high-order intuitionistic logic on (quantum) states. In this context, the programming language of quantum channel construction equipped with equation theory (given by the equality of denotation) might be transformed into the programming language of quantum states equipped with high-order logic (whose type system can represent any formula in the logic).

This approach looks nice since it could solve the problem of inductive type and recursion. The type system can define an inductive type like a natural number and the primitive recursion as the induction principle from the level of logic. Moreover, we could define some properties of quantum states by introducing an additional set of typing derivation rules corresponding to a first-order theory that defines the property. However, it is not clear if this approach since it is not yet clear what exactly is the lifted monad over the presheaf. Nevertheless, it would be nice to check this approach properly since it might give a computational meaning to the topos of quantum states.

Bibliography

- [1] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004., pages 415–425, 2004. doi: 10.1109/LICS.2004.1319636.
- [2] R. M. Amadio and P.-L. Curien. Domains and Lambda-Calculi, volume 46 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2008.
- [3] ANSI INCITS 226-1994. The common lisp standard. Technical Report ANSI INCITS 226-1994, ANSI, 1994.
- [4] L. Anticoli, C. Piazza, L. Taglialegne, and P. Zuliani. Towards quantum programs verification: from quipper circuits to qpmc. In International Conference on Reversible Computation, pages 213–219. Springer, 2016.
- [5] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, Oct 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1666-5. URL <https://doi.org/10.1038/s41586-019-1666-5>.
- [6] H. Barendregt, W. Dekkers, and R. Statman. Lambda Calculus with Types. Perspective in Logic. Cambridge University Press and ASL, 2013.
- [7] H. P. Barendregt. The Lambda-Calculus, its Syntax and Semantics, volume 103 of Studies in Logic and the Foundation of Mathematics. North Holland, 2 edition, 1984. ISBN 0-444-86748-1.

- [8] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, 2014. ISSN 0304-3975. doi: <https://doi.org/10.1016/j.tcs.2014.05.025>. URL <https://www.sciencedirect.com/science/article/pii/S0304397514004241>. *Theoretical Aspects of Quantum Cryptography – celebrating 30 years of BB84*.
- [9] N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, Eighth International Workshop, CSL’94, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. European Association for Computer Science Logic, Springer Verlag, September 1994. ISBN 3-540-60017-5. doi: 10.1007/BFb0022251.
- [10] N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. Technical Report UCAM-CL-TR-352, Computer Science department, Cambridge University, 1994. 65 pages.
- [11] N. Benton, G. M. Bierman, M. Hyland, and V. C. V. de Paiva. Linear lambda-calculus and categorical models revisited. In E. Börger, G. Jäger, H. K. Büning, S. Martini, and M. M. Richter, editors, *Computer Science Logic, Sixth International Workshop, CSL’92, Selected Papers*, volume 702 of *Lecture Notes in Computer Science*. European Association for Computer Science Logic, Springer Verlag, September 1992. ISBN 978-3-540-56992-3. doi: 10.1007/3-540-56992-8_6.
- [12] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *International Workshop on Computer Science Logic*, pages 121–135. Springer, 1994.
- [13] G. M. Bierman. *On Intuitionistic Linear Logic*. PhD thesis, Computer Science department, Cambridge University, England, UK., 1993. Available as Technical Report 346, August 1994.
- [14] G. M. Bierman. What is a categorical model of intuitionistic linear logic. In M. Dezani-Ciancaglini and G. D. Plotkin, editors, *Proceedings of the Second International Conference on Typed Lambda Calculi and Applications, TLCA’95*, volume 902 of *Lecture Notes in Computer Science*, pages 78–93. Springer Verlag, April 1995. ISBN 3-540-59048-X. doi: 10.1007/BFb0014046.
- [15] W. Brinkman, D. Haggan, and W. Troutman. A history of the invention of the transistor and where it will lead us. *IEEE Journal of Solid-State Circuits*, 32(12):1858–1865, 1997. doi: 10.1109/4.643644.

- [16] B. Coecke and A. Kissinger, editors. Proceedings 14th International Conference on Quantum Physics and Logic, QPL 2017, volume 266 of Electronic Proceedings in Theoretical Computer Science, 2018.
- [17] H. B. Curry. Functionality in combinatory logic. Proceedings of the National Academy of Sciences, 20(11):584–590, 1934. ISSN 0027-8424. doi: 10.1073/pnas.20.11.584.
- [18] H. H. Curry, R. Feys, and W. Craig. Combinatory Logic, volume 22 of Studies in Logic and the Foundations of Mathematics. North-Holland, 1958.
- [19] D. Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences, 400(1818):97–117, 1985.
- [20] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences, 439(1907):553–558, 1992.
- [21] D. Doligez, A. Frisch, J. Garrigue, D. Rémy, and J. Vouillon. The ocaml system: Documentation and user’s manual. Online reference¹.
- [22] T. Ehrhard, J.-Y. Girard, P. Ruet, and P. Scott, editors. Linear Logic in Computer Science, volume 316 of London Mathematical Society Lecture Notes Series. Cambridge University Press, 2004. ISBN 0-521-60857-0.
- [23] P. Fu, K. Kishida, N. J. Ross, and P. Selinger. A tutorial introduction to quantum circuit programming in dependently typed proto-quipper. In I. Lanese and M. Rawski, editors, Proceedings of the 12th International Conference on Reversible Computation, RC 2020, volume 12227 of Lecture Notes in Computer Science, pages 153–168. Springer, 2020. ISBN 978-3-030-52481-4. doi: 10.1007/978-3-030-52482-1_9.
- [24] P. Fu, K. Kishida, and P. Selinger. Linear dependent type theory for quantum programming languages: Extended abstract. In H. Hermanns, L. Zhang, N. Kobayashi, and D. Miller, editors, Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2020, pages 440–453. ACM, July 2020. ISBN 978-1-4503-7104-9. doi: 10.1145/3373718.3394765.
- [25] J.-Y. Girard. Linear logic. Theoretical Computer Science, 50(1):1–101, 1987. doi: 10.1016/0304-3975(87)90045-4.

¹<https://ocaml.org/releases/latest/manual.html>

- [26] J.-Y. Girard, Y. Lafont, and L. Regnier, editors. *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1995. ISBN 0-521-55961-8.
- [27] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. An introduction to quantum programming in quipper. In G. W. Dueck and D. M. Miller, editors, *Proceedings of the 5th International Conference on Reversible Computation, RC'13*, volume 7948 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2013. ISBN 978-3-642-38985-6. doi: 10.1007/978-3-642-38986-3_10.
- [28] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *ACM SIGPLAN Notices*, volume 48, pages 333–342. ACM, 2013.
- [29] W. Heisenberg. The development of quantum mechanics, pages 226–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 1984. ISBN 978-3-642-61742-3. doi: 10.1007/978-3-642-61742-3_15. URL https://doi.org/10.1007/978-3-642-61742-3_15.
- [30] C. Heunen, N. P. Landsman, and B. Spitters. A topos for algebraic quantum theory. *Communications in mathematical physics*, 291(1):63–110, 2009.
- [31] J. D. Hidary. *A Brief History of Quantum Computing*, pages 15–21. Springer International Publishing, Cham, 2021. ISBN 978-3-030-83274-2. doi: 10.1007/978-3-030-83274-2_2. URL https://doi.org/10.1007/978-3-030-83274-2_2.
- [32] W. A. Howard. The formulae-as-type notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.
- [33] S. P. Jones, editor. *Haskell 98 language and libraries: the Revised Report*. Cambridge University Press, 2003.
- [34] E. Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Lab., NM (United States), 1996.
- [35] E. H. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, Los Alamos, New Mexico, US., 1996.
- [36] J. Lambek and P. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Cambridge studies in advanced mathematics*. Cambridge University Press, 2 edition, 1989. ISBN 0-521-35653-9.

- [37] R. H. Landau, J. Páez, and C. C. Bordeianu. *A Survey of Computational Physics: Introductory Computational Science*. Princeton University Press, 2011. ISBN 9781400841189. doi: [doi:10.1515/9781400841189](https://doi.org/10.1515/9781400841189). URL <https://doi.org/10.1515/9781400841189>.
- [38] B. Lindenhovius, M. W. Mislove, and V. Zamdzhiev. Enriching a linear/non-linear lambda calculus: A programming language for string diagrams. In A. Dawar and E. Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'2018*, pages 659–668. ACM, 2018. doi: [10.1145/3209108.3209196](https://doi.org/10.1145/3209108.3209196).
- [39] B. Lindenhovius, M. Mislove, and V. Zamdzhiev. *Mixed linear and non-linear recursive types*. volume 3, New York, NY, USA, 2019. ICFP, Association for Computing Machinery.
- [40] O. Malherbe. *Categorical models of computation: partially traced categories and presheaf models of quantum computation*. PhD thesis, University of Ottawa, 2010.
- [41] O. Malherbe, P. Scott, and P. Selinger. Presheaf models of quantum computation: An outline. In B. Coecke, L. Ong, and P. Panangaden, editors, *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky – Essays Dedicated to Samson Abramsky on the Occasion of His 60th Birthday*, volume 7860 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2013. doi: [10.1007/978-3-642-38164-5_13](https://doi.org/10.1007/978-3-642-38164-5_13).
- [42] E. Moggi. *Computational lambda-calculus and monads*. Technical Report ECS-LFCS-88-66, Laboratory for Foundation of Computer Sciences, Edinburgh University, Scotland, UK., 1988. URL <http://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-66/>.
- [43] E. Moggi. *Computational lambda-calculus and monads*. In *Proceedings of the Fourth Symposium on Logic in Computer Science, LICS'89*, pages 14–23. IEEE, IEEE Computer Society Press, June 1989. ISBN 0-8186-1954-6. doi: [10.1109/LICS.1989.39155](https://doi.org/10.1109/LICS.1989.39155). URL <http://www.lfcs.inf.ed.ac.uk/reports/88/ECS-LFCS-88-66/>.
- [44] E. Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.
- [45] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002. ISBN 0-521-63503-9.

- [46] M. Pagani, P. Selinger, and B. Valiron. Applying quantitative semantics to higher-order quantum computing. In *ACM SIGPLAN Notices*, volume 49, pages 647–658. ACM, 2014.
- [47] J. Paykin, R. Rand, and S. Zdancewic. Qwire: A core language for quantum circuits. In *ACM SIGPLAN Notices*, volume 52, pages 846–858. ACM, 2017.
- [48] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1(2):125–159, 1975. doi: 10.1016/0304-3975(75)90017-1.
- [49] J. M. Raimond and S. Haroche. Quantum Computing: Dream Or Nightmare. In *31st Rencontres de Moriond: Dark Matter and Cosmology, Quantum Measurements and Experimental Gravitation*, pages 341–346, Gif-Sur-Yvette, 1996. Ed. Frontieres.
- [50] R. Rand, J. Paykin, and S. Zdancewic. QWIRE practice: Formal verification of quantum circuits in coq. In Coecke and Kissinger [16], pages 119–132. doi: 10.4204/EPTCS.266.8.
- [51] R. Rand, J. Paykin, D. Lee, and S. Zdancewic. Reqwire: Reasoning about reversible quantum circuits. In P. Selinger and G. Chiribella, editors, *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018*, volume 287 of *EPTCS*, pages 299–312, 2018. doi: 10.4204/EPTCS.287.17.
- [52] R. Rand, J. Paykin, and S. Zdancewic. Qwire practice: Formal verification of quantum circuits in coq. arXiv preprint arXiv:1803.00699, 2018.
- [53] M. Rennela and S. Staton. Classical control and quantum circuits in enriched category theory. *Electronic Notes in Theoretical Computer Science*, 336:257–279, 2018.
- [54] H. Riel. Quantum computing technology. In *2021 IEEE International Electron Devices Meeting (IEDM)*, pages 1.3.1–1.3.7, 2021. doi: 10.1109/IEDM19574.2021.9720538.
- [55] F. Rios and P. Selinger. A categorical model for a quantum circuit description language. In Coecke and Kissinger [16], pages 164–178. doi: 10.4204/EPTCS.266.11.
- [56] N. J. Ross. Algebraic and logical methods in quantum computation. PhD thesis, 2015.

- [57] P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004. doi: [10.1017/S0960129504004256](https://doi.org/10.1017/S0960129504004256).
- [58] P. Selinger. Towards a semantics for higher-order quantum computation. In P. Selinger, editor, *Proceedings of the Second International Workshop on Quantum Programming Languages, QPL'04*, volume 33 of *TUCS General Publication*, pages 127–143. TUCS, 2004. ISBN 9-5212-1374-4. URL <http://urn.fi/URN:NBN:fi:bib:me:I00035039900>.
- [59] P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.
- [60] P. Selinger. Dagger compact closed categories and completely positive maps: (extended abstract). *Electronic Notes in Theoretical Computer Science*, 170:139–163, 2007. ISSN 1571-0661. doi: <https://doi.org/10.1016/j.entcs.2006.12.018>. URL <https://www.sciencedirect.com/science/article/pii/S1571066107000606>. *Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005)*.
- [61] P. Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.
- [62] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. In P. Urzyczyn, editor, *Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications, TLCA'05*, volume 3461 of *Lecture Notes in Computer Science*, pages 354–368. Springer Verlag, April 2005. ISBN 3-540-25593-1. doi: [10.1007/11417170_26](https://doi.org/10.1007/11417170_26). Journal version appeared in *MSCS* [63].
- [63] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.
- [64] P. Selinger and B. Valiron. On a fully abstract model for a quantum linear functional language. In P. Selinger, editor, *Proceedings of the Fourth International Workshop on Quantum Programming Languages, QPL'06*, volume 210 of *Electronic Notes in Theoretical Computer Science*, pages 123–137, July 2008. doi: [10.1016/j.entcs.2008.04.022](https://doi.org/10.1016/j.entcs.2008.04.022).
- [65] P. Selinger and B. Valiron. Quantum lambda-calculus. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*, chapter 4, pages 135–172. Cambridge University Press, 2009. ISBN 978-0-521-51374-6.

- [66] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th Annual Symposium on Foundations of Computer Science, pages 124–134, 1994. doi: 10.1109/SFCS.1994.365700.
- [67] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493–R2496, Oct 1995. doi: 10.1103/PhysRevA.52.R2493. URL <https://link.aps.org/doi/10.1103/PhysRevA.52.R2493>.
- [68] J. M. Smith, N. J. Ross, P. Selinger, and B. Valiron. Quipper: Concrete resource estimation in quantum algorithms. In Informal Proceedings of QAPL’14, Grenoble, France, 2014.
- [69] R. S. Smith, M. J. Curtis, and W. J. Zeng. A practical quantum instruction set architecture. arXiv preprint arXiv:1608.03355, 2016.
- [70] S. Staton. Algebraic effects, linearity, and quantum programming languages. *SIGPLAN Not.*, 50(1):395–406, Jan. 2015. ISSN 0362-1340. doi: 10.1145/2775051.2676999. URL <http://doi.acm.org/10.1145/2775051.2676999>.
- [71] A. Stoughton. Fully Abstract Models of Programming Languages. Research Notes in Computer Science. Pitman Publishing, Inc., Marshfield, Massachusetts, US., 1988. ISBN 0-470-21041-9. Revised version of the author’s Ph.D. thesis from the University of Edinburgh, accessible online at <http://hdl.handle.net/1842/14496>.
- [72] K. M. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler. Q#: Enabling scalable quantum computing and development with a high-level domain-specific language. arXiv preprint arXiv:1803.00652, 2018.
- [73] A. S. Troelstra. Lectures in Linear Logic, volume 29 of CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, California, US., 1992. ISBN 0-937073-77-6.
- [74] B. Valiron. A Functional Programming Language for Quantum Computation With Classical Control. Master thesis, University of Ottawa, 2004.
- [75] B. Valiron. Semantics for a Higher Order Functional Programming Language for Quantum Computation. PhD thesis, University of Ottawa, 2008.

- [76] B. Valiron. Semantics for a higher order functional programming language for quantum computation. PhD thesis, University of Ottawa, 2008.
- [77] D. Wecker and K. M. Svore. LIQUi|>: A software design architecture and domain-specific language for quantum computing. arXiv preprint arXiv:1402.4467, 2014.
- [78] A. Westerbaan. Quantum programs as kleisli maps. In R. Duncan and C. Heunen, editors, Proceedings of the 13th International Conference on Quantum Physics and Logic, QPL 2016, volume 236 of Electronic Proceedings in Theoretical Computer Science, pages 215–228, 2016. doi: 10.4204/EPTCS.236.14.
- [79] A. A. Westerbaan. The Category of Von Neumann Algebras. PhD thesis, Radboud Universiteit Nijmegen, 2019. URL <https://hdl.handle.net/2066/201611>.
- [80] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. Nature, 299:802–803, October 1982. doi: 10.1038/299802a0.
- [81] B. Ömer. Structured quantum programming. PhD thesis, 2003.

A - Proof of the Preservation Theorem

This contains the proof of Theorem 5.3.5.

If $(Q_1, m_1) \rightarrow (Q_2, m_2)$, then
 for all π derivation of $\vdash(Q_1, m_1):A$, there exists a
 derivation π' of $\vdash(Q_2, m_2):A$ st
 $\llbracket \pi \rrbracket \vdash(Q_1, m_1):A \rrbracket = \llbracket \pi' \rrbracket \vdash(Q_2, m_2):A \rrbracket$.

The proof is done by induction on the derivation of the reduction. To ease legibility, each case of the induction is presented in its own section.

Table of contents for the proof

A.1 Beta-reduction	282
A.2 Let-product elimination	287
A.3 If-then-else	294
A.4 Box	299
A.5 Unbox	305
A.6 Congruence: quantum channel	328
A.7 Congruence: on the right of application	331
A.8 Congruence: on the left of application	356
A.9 Congruence: on the left of a pair	373
A.10 Congruence: on the right of a pair	391
A.11 Congruence: if-then-else	407
A.12 Congruence: let-construct	422
A.13 Congruence: branching, case 1	442
A.14 Congruence: branching, case 2	447
A.15 Congruence: branching, case 3	452
A.16 Congruence: circuit reduction	457

A.1 . Beta-reduction

$$\overline{(\varepsilon(w), (\lambda x.M)V) \rightarrow (\varepsilon(w), M[V/x])}$$

$$\llbracket \vdash (\varepsilon(w), (\lambda x.M)V) : A \rrbracket = \llbracket (\llbracket \varepsilon(w) \rrbracket, f) \rrbracket$$

$$\llbracket \vdash (\varepsilon(w), M[V/x]) : A \rrbracket = \llbracket (\llbracket \varepsilon(w) \rrbracket, g) \rrbracket$$

$$f = \llbracket \nu \text{Bind}(\phi, w, (\lambda x.M)V, A) \rrbracket$$

$$= \llbracket w : \text{qubit} \vdash (\lambda x.M)V : A \rrbracket$$

$$g = \llbracket \nu \text{Bind}(\phi, w, M[V/x], A) \rrbracket$$

$$= \llbracket w : \text{qubit} \vdash M[V/x] : A \rrbracket$$

$$\left(FV((\lambda x.M)V) = FV(M[V/x]) \right)$$

(\because by definition)

② $f = g$ (Lemma (substitution).)

from $w: \text{qubit} \vdash (\lambda x. M) V : A$,

$$\frac{\frac{Q_a: \text{qubit}, x: A_a \vdash M : A}{Q_a: \text{qubit} \vdash (\lambda x. M) : A_a \rightarrow A} \quad (W = Q_a \otimes Q_b) \quad Q_b: \text{qubit} \vdash V : A_a}{w: \text{qubit} \vdash (\lambda x. M) V : A}$$

Let $h_a = \llbracket Q_a, x: A_a \vdash M : A \rrbracket$
 $h_b = \llbracket Q_b \vdash V : A_a \rrbracket = \llbracket Q_b \rrbracket \xrightarrow{h'_a} \llbracket A_a \rrbracket \xrightarrow{\eta} F \llbracket A_a \rrbracket$

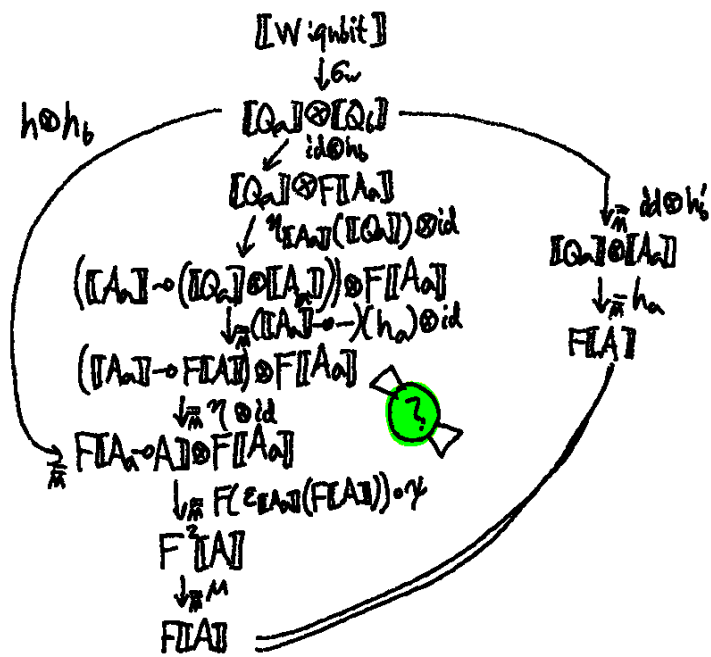
By definition,

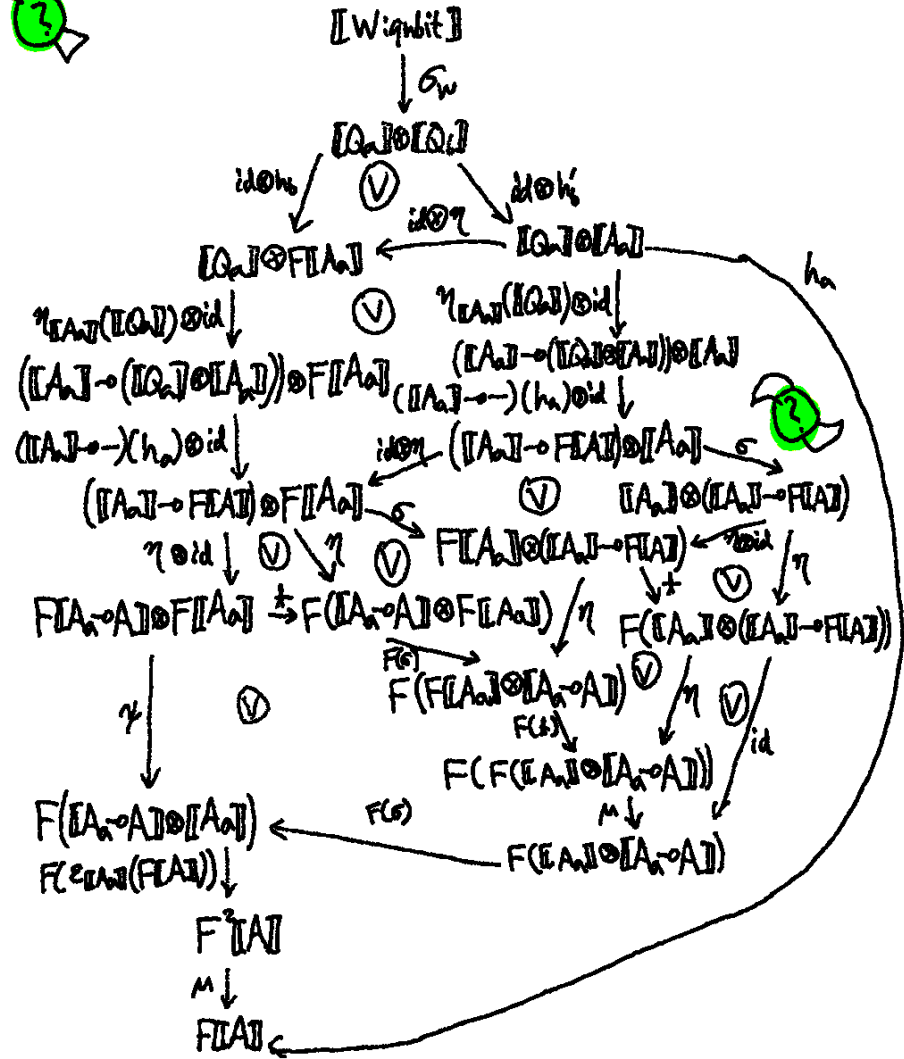
$$\begin{aligned} f &= \llbracket w: \text{qubit} \vdash (\lambda x. M) V : A \rrbracket \\ &= \llbracket w: \text{qubit} \rrbracket \xrightarrow{\Sigma} \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \xrightarrow[\text{id}]{(\text{id} \otimes h'_a) \cdot (\text{id} \otimes \eta)} \llbracket Q_a \rrbracket \otimes \llbracket Q_b \rrbracket \\ &\quad \downarrow h \circ h_b \\ &\quad F \llbracket A_a \rightarrow A \rrbracket \otimes F \llbracket A_a \rrbracket \\ &\quad \downarrow F(\text{id} \otimes \eta)(F \llbracket A_a \rrbracket) \circ \eta \\ &\quad F \llbracket A \rrbracket \\ &\quad \downarrow M \\ &\quad F \llbracket A \rrbracket \end{aligned}$$

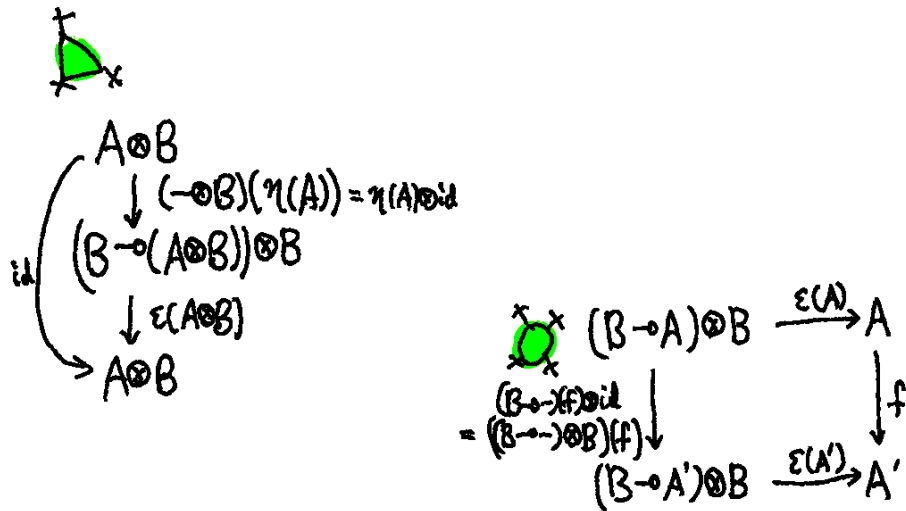
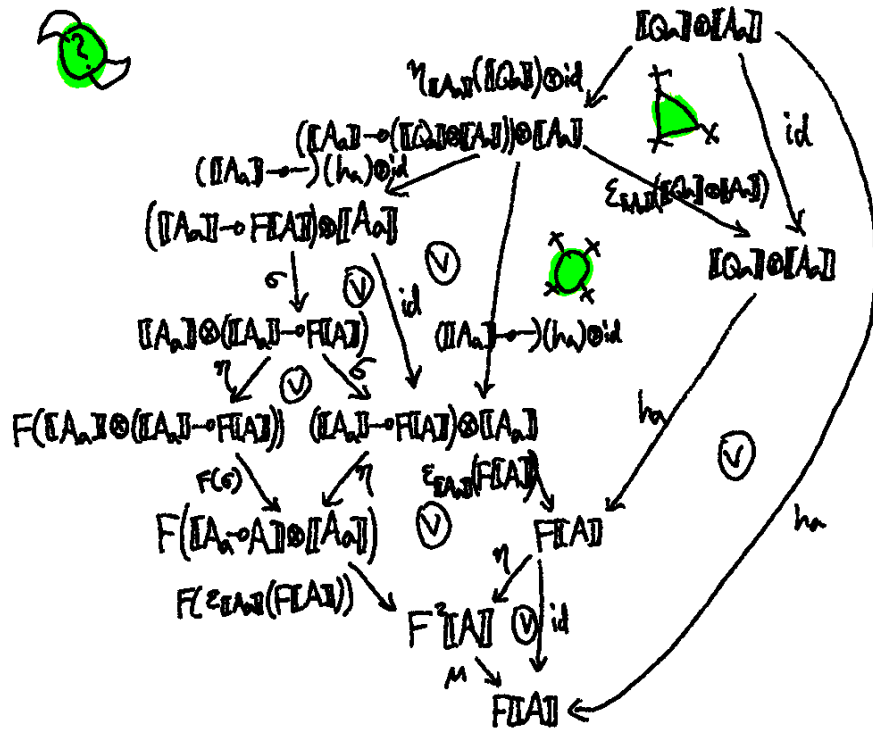
$$\begin{aligned} h &= \llbracket Q_a: \text{qubit} \vdash (\lambda x. M) : A_a \rightarrow A \rrbracket \\ &= \llbracket Q_a \rrbracket \xrightarrow{\llbracket A_a \rrbracket \circ (\llbracket Q_a \rrbracket)} \llbracket A_a \rrbracket \circ (\llbracket Q_a \rrbracket \otimes \llbracket A_a \rrbracket) \\ &\quad \downarrow \llbracket A_a \rrbracket \circ - \circ \chi(h_a) \\ &\quad \llbracket A_a \rrbracket \rightarrow F \llbracket A \rrbracket \\ &\quad \downarrow \eta \\ &\quad F \llbracket A_a \rightarrow A \rrbracket \end{aligned}$$

By substitution lemma,

$$\begin{aligned}
 g &= \llbracket W:\text{qubit} \vdash M[V/x] : A_6 \rrbracket \\
 &= \llbracket W:\text{qubit} \rrbracket \xrightarrow{\sigma_v} \llbracket Q_n \rrbracket \otimes \llbracket Q_d \rrbracket \xrightarrow{h \otimes h'} \llbracket Q_n \rrbracket \otimes \llbracket A_n \rrbracket \xrightarrow{h_n} \llbracket F[A] \rrbracket
 \end{aligned}$$







A.2 . Let-product elimination

$$\begin{aligned} & \overline{(\varepsilon(W), \text{let } \langle x, y \rangle = \langle V, U \rangle \text{ in } M) \rightarrow (\varepsilon(W), M[V/x, U/y])} \\ & \llbracket \vdash (\varepsilon(W), \text{let } \langle x, y \rangle = \langle V, U \rangle \text{ in } M) : A \rrbracket = \llbracket \llbracket \varepsilon(W) \rrbracket, f \rrbracket \\ & \llbracket \vdash (\varepsilon(W), M[V/x, U/y]) : A \rrbracket = \llbracket \llbracket \varepsilon(W) \rrbracket, g \rrbracket \\ & f = \llbracket W : \text{qubit} \vdash \text{let } \langle x, y \rangle = \langle V, U \rangle \text{ in } M : A \rrbracket \\ & g = \llbracket W : \text{qubit} \vdash M[V/x, U/y] : A \rrbracket \\ & (FV(M[V/x, U/y]) = FV(\text{let } \langle x, y \rangle = \langle V, U \rangle \text{ in } M)) \end{aligned}$$

By the α -equivalence, we can assume w.l.g.
 $M[V/x, U/y] = M[V/x][U/y] = M[U/y][V/x]$
 by letting $x, y \notin FV(V) \cup FV(U)$.

$$\begin{array}{c} \frac{Q_a^1 \vdash V : A_a \quad Q_a^2 \vdash U : A_b \quad (W = Q_a U Q_b, Q_a = Q_a^1 Q_a^2)}{Q_a : \text{qubit} \vdash \langle V, U \rangle : A_a \otimes A_b \quad Q_b : \text{qubit}, x : A_a, y : A_b \vdash M : A} \\ W : \text{qubit} \vdash \text{let } \langle x, y \rangle = \langle V, U \rangle \text{ in } M : A \\ \text{Let } h_a = \llbracket Q_a^1 \vdash V : A_a \rrbracket \quad h_b = \llbracket Q_a^2 \vdash U : A_b \rrbracket \\ h_{\otimes} = \llbracket Q_a : \text{qubit} \vdash \langle V, U \rangle : A_a \otimes A_b \rrbracket \\ h = \llbracket Q_b : \text{qubit}, x : A_a, y : A_b \vdash M : A \rrbracket \end{array}$$

By the substitution lemma:

$$\begin{aligned}
 h_c &= [\!| Q_0 : \text{qubit}, x : A_n, Q_n^2 \vdash M[U/y] : A \!|] \\
 &= [\!| \text{qubit} \!| \otimes [\!| A_n \!| \otimes [\!| Q_n^2 \!|] \\
 &\quad \downarrow_{\bar{\pi}} \text{id} \otimes h_c^\circ \\
 &= [\!| \text{qubit} \!| \otimes [\!| A_n \!| \otimes [\!| A_n \!|] \\
 &\quad \downarrow_{\bar{\pi}} h \\
 &= F[\!| A \!|]
 \end{aligned}$$


$$\begin{aligned}
 g' &= [\!| Q_0 : \text{qubit}, Q_n^2, Q_n^1 \vdash M[U/y][V/h] : A \!|] \\
 &= [\!| \text{qubit} \!| \otimes [\!| Q_n^2 \!| \otimes [\!| Q_n^1 \!|] \\
 &\quad \downarrow_{\bar{\pi}} \text{id} \otimes h_n^\circ \\
 &= [\!| \text{qubit} \!| \otimes [\!| Q_n^2 \!| \otimes [\!| A_n \!|] \\
 &\quad \downarrow_{\bar{\pi}} h_c \circ \sigma_h \\
 &= F[\!| A \!|]
 \end{aligned}$$

$$g = g' \circ \sigma_g$$

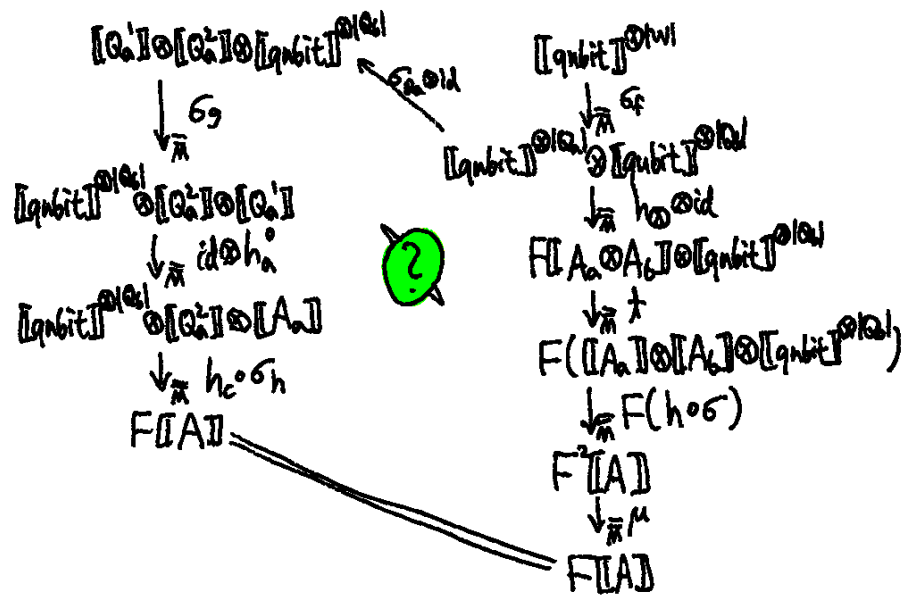
By definition:

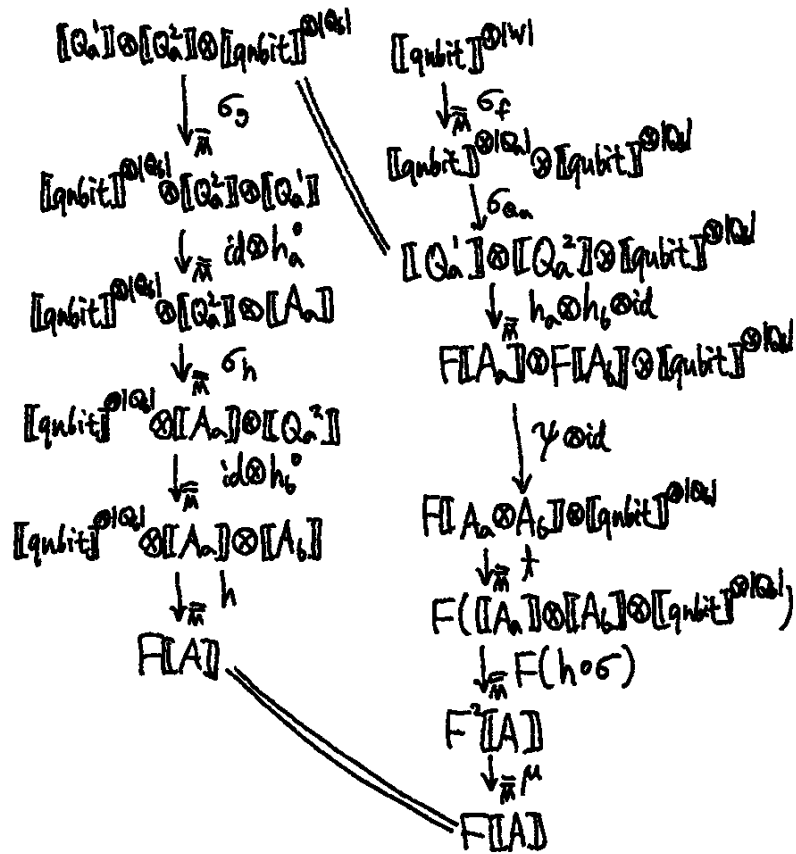
$$\begin{aligned}
 h_0 &= [Q_a \text{ qubit} \langle V/U \rangle: A_n \otimes A_b] \\
 &= [Q_n^1] \otimes [Q_n^2] \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}}(id \otimes id) \circ (d_1 \otimes id)} = id \\
 &[Q_n^1] \otimes [Q_n^2] \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}} h_n \otimes h_b} \\
 &F[A_n] \otimes F[A_b] \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}} \gamma} \\
 &F[A_n \otimes A_b]
 \end{aligned}$$

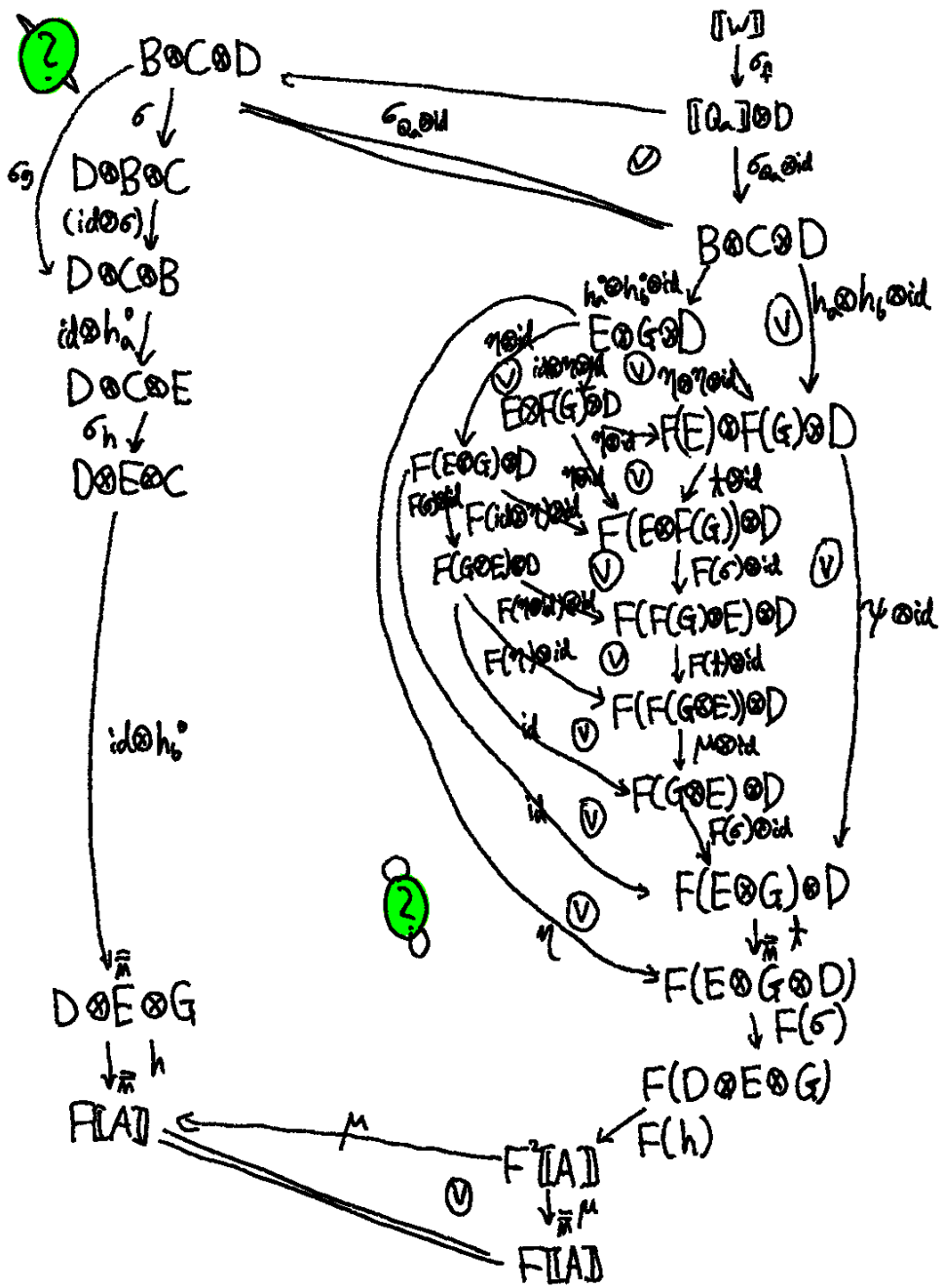
$$\begin{aligned}
 f &= [qubit]^{\otimes n} \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}}(id \otimes id) \circ (d_1 \otimes id)} = G_r \\
 &[qubit]^{\otimes n} \otimes [qubit]^{\otimes n} \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}} h_n \otimes id} \\
 &F[A_n \otimes A_b] \otimes [qubit]^{\otimes n} \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}} \dagger} \\
 &F([A_n] \otimes [A_b] \otimes [qubit]^{\otimes n}) \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}} F(h \circ \sigma)} \\
 &F^2[A] \\
 &\quad \downarrow_{\frac{1}{\sqrt{2}} \mu} \\
 &F[A]
 \end{aligned}$$

 We show the following commutes:

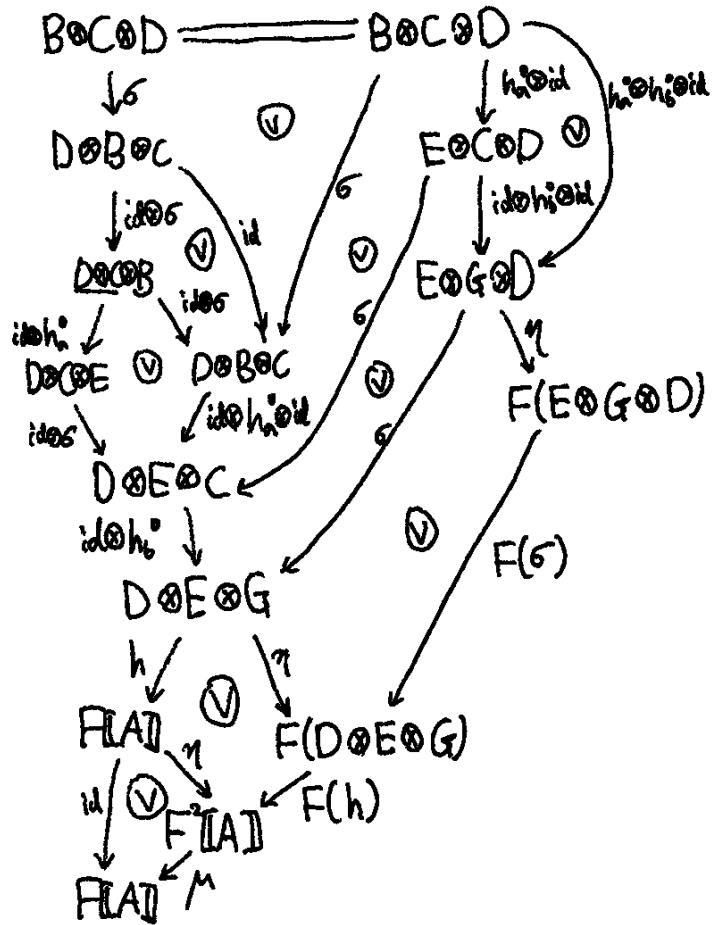
$$(f=g)$$







2



A.3 . If-then-else

$$\begin{aligned} & \overline{(\varepsilon(w), \text{if } tt \text{ then } M_a \text{ else } M_b) \rightarrow (\varepsilon(w), M_a)} \\ & \llbracket !\Delta \vdash (\varepsilon(w), \text{if } tt \text{ then } M_a \text{ else } M_b) : A \rrbracket \\ & \quad = \llbracket (\llbracket \varepsilon(w) \rrbracket, f) \rrbracket \\ & \llbracket !\Delta \vdash (\varepsilon(w), M_a) : A \rrbracket = \llbracket (\llbracket \varepsilon(w) \rrbracket, g) \rrbracket \\ & f = \llbracket !\Delta, w : \text{qubit} \vdash \text{if } tt \text{ then } M_a \text{ else } M_b : A \rrbracket \\ & g = \llbracket !\Delta, w : \text{qubit} \vdash M_a : A \rrbracket \end{aligned}$$

Since

$$\frac{!\Delta \vdash tt : \text{bool} \quad !\Delta, w : \text{qubit} \vdash M_a : A \quad !\Delta, w : \text{qubit} \vdash M_b : A}{!\Delta, w : \text{qubit} \vdash \text{if } tt \text{ then } M_a \text{ else } M_b : A}$$

$$\text{Let } h_{tt} = \llbracket !\Delta \vdash tt : \text{bool} \rrbracket, \quad h_a = \llbracket !\Delta, w : \text{qubit} \vdash M_a : A \rrbracket \\ h_b = \llbracket !\Delta, w : \text{qubit} \vdash M_b : A \rrbracket$$

By definition: $g = h_a$ and

$$f = [! \Delta, \text{wqubit} \text{ if } tt \text{ then } M_a \text{ else } M_b : A]$$

$$= [! \Delta] \otimes [qubit]^{otw}$$

$$\downarrow_{\overline{M}} (\text{id} \otimes \text{id}) \circ (\text{id} \otimes \text{id})$$

$$[! \Delta] \otimes [! \Delta] \otimes [qubit]^{otw}$$

$$\downarrow_{\overline{M}} h_{tt} \otimes \text{id}$$

$$F[\text{bool}] \otimes [! \Delta] \otimes [qubit]^{otw}$$

$$\downarrow_{\overline{M}} \dagger$$

$$F([! \Delta] \otimes [qubit]^{otw})$$

$$\downarrow_{\overline{M}} F(\star)$$


$$F([! \Delta] \otimes [qubit]^{otw} + [! \Delta] \otimes [qubit]^{otw})$$

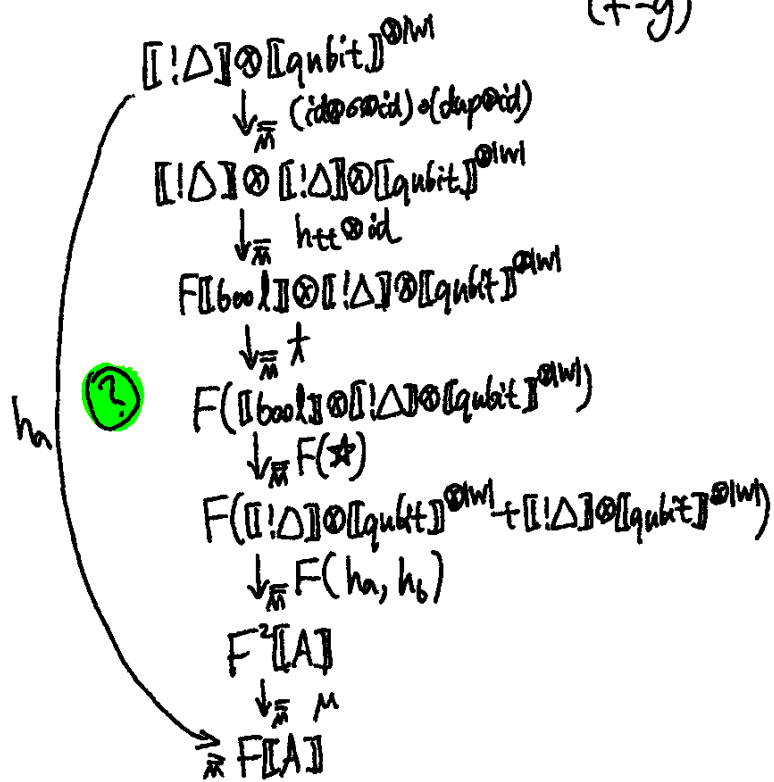
$$\downarrow_{\overline{M}} F(h_a, h_b)$$

$$F^2[A]$$

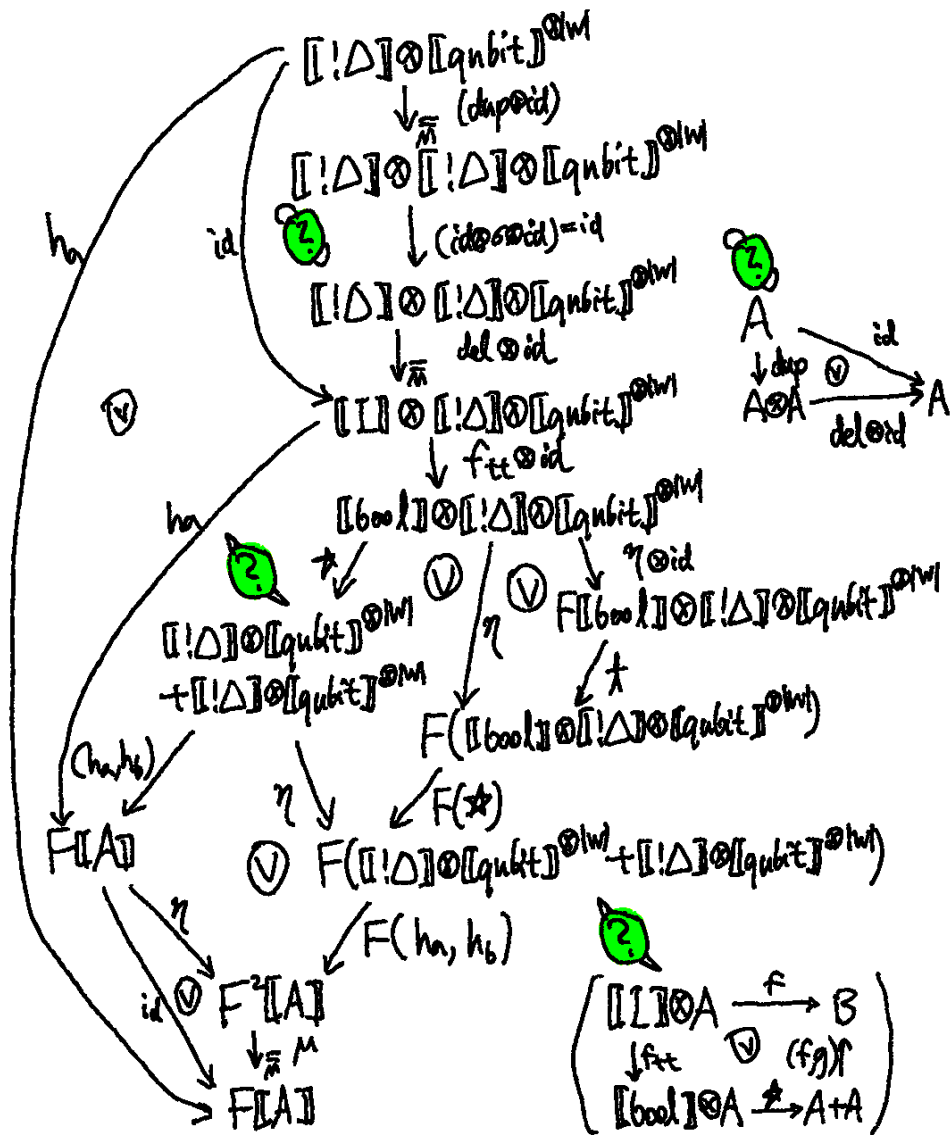
$$\downarrow_{\overline{M}} M$$

$$F[A]$$

 We show the following commutes: $(f=g)$



Ⓜ We show the following commutes:



$(\mathcal{E}(W), \text{if } ff \text{ then } M_a \text{ else } M_b) \rightarrow (\mathcal{E}(W), M_b)$

(Similar to the previous case.)

A.4 . Box

$$\frac{\rho = \text{new}(P) \quad w_p = \text{supp}(P)}{(\varepsilon(\phi), \text{box}_p V) \rightarrow (\varepsilon(\phi), (p, \varepsilon(w_p), V_p))}$$

$$\llbracket \vdash (\varepsilon(\phi), \text{box}_p V) : A' \rrbracket = \llbracket (\llbracket \varepsilon(\phi) \rrbracket, f) \rrbracket$$

$$\llbracket \vdash (\varepsilon(\phi), (p, \varepsilon(w_p), V_p)) : A' \rrbracket = \llbracket (\llbracket \varepsilon(\phi) \rrbracket, g) \rrbracket$$

($A' = !Q\text{Chan}(P, A)$)

$$f = \llbracket \vdash \text{box}_p V : A' \rrbracket$$

$$g = \llbracket \vdash (p, \varepsilon(w_p), V_p) : A' \rrbracket$$

$$\frac{\vdash \text{box}_p : !(P \multimap A) \multimap !Q\text{Chan}(P, A) \quad \vdash V : !(P \multimap A)}{\vdash \text{box}_p V : !Q\text{Chan}(P, A)}$$

$$\frac{\rho = P \quad v\text{Bind}(\phi, w_p, V_p, A)}{\vdash (p, \varepsilon(w_p), V_p) : !Q\text{Chan}(P, A)}$$

(The other case where $A' = Q\text{Chan}(P, A)$ is the same except for the dereliction on both sides.)

By definition:

$$\begin{aligned}
 f &= \llbracket \vdash \text{box}_p V : !Q\text{chan}(P, A) \rrbracket \\
 &= I \xrightarrow{(id \otimes \text{id}) \circ (\text{dup} \otimes id)} I \otimes I \\
 &\quad \downarrow_{\bar{m}} f_{\text{box}} \otimes f_V \\
 &\quad F[\llbracket (P \multimap A) \multimap !Q\text{chan}(P, A) \rrbracket \\
 &\quad \otimes F[\llbracket (P \multimap A) \rrbracket] \\
 &\quad \downarrow_{\bar{m}} \eta \\
 &\quad F[\llbracket (P \multimap A) \multimap !Q\text{chan}(P, A) \rrbracket \\
 &\quad \otimes \llbracket (P \multimap A) \rrbracket] \\
 &\quad F(\varepsilon_{!P \multimap A} (F[\llbracket !Q\text{chan}(P, A) \rrbracket])) \downarrow_{\bar{m}} \\
 &\quad F^2[\llbracket !Q\text{chan}(P, A) \rrbracket] \\
 &\quad \downarrow_{\bar{m}} \eta \\
 &\quad F[\llbracket !Q\text{chan}(P, A) \rrbracket]
 \end{aligned}$$

where

$$\begin{aligned}
 f_V &= \llbracket \vdash V : !(P \multimap A) \rrbracket \\
 &= I \xrightarrow{f_V^0} \llbracket !(P \multimap A) \rrbracket \xrightarrow{\eta} F[\llbracket (P \multimap A) \rrbracket] \quad \left(\begin{array}{l} \text{by the lemma on duplicate} \\ \text{values, } f_V^0 = f^{(1, f_V)} \end{array} \right)
 \end{aligned}$$

$$\begin{aligned}
 f_{\text{box}} &= \llbracket \vdash \text{box}_p : !(P \multimap A) \multimap !Q\text{chan}(P, A) \rrbracket \\
 &= I \xrightarrow{\text{del}} I \xrightarrow{\eta_{!P \multimap A}(I)} \llbracket !P \multimap A \rrbracket \multimap (I \otimes \llbracket !P \multimap A \rrbracket) \\
 &\quad \downarrow_{\bar{m}} (\llbracket !P \multimap A \rrbracket \multimap \text{box}) \\
 &\quad \llbracket !P \multimap A \rrbracket \multimap F[\llbracket !Q\text{chan}(P, A) \rrbracket] \\
 &\quad \downarrow_{\bar{m}} \eta \\
 &\quad F[\llbracket (P \multimap A) \multimap !Q\text{chan}(P, A) \rrbracket]
 \end{aligned}$$

Again by definition,

$$g = \llbracket \vdash (p, \varepsilon(w_p), V_p) : !\text{Chan}(p, A) \rrbracket$$

$$= \downarrow_{\bar{\alpha}} \frac{\pi^* \cdot \delta^{\otimes 2}}{\pi^*} \llbracket \downarrow_{\bar{\alpha}} !(\eta(I)) \rrbracket$$

$$\downarrow_{\bar{\alpha}} \llbracket !(\text{IP} \rightarrow \text{I} \otimes \text{PI}) \rrbracket$$

$$\downarrow_{\bar{\alpha}} \llbracket !(\text{IP} \rightarrow \text{I}) \rrbracket (h)$$

$$\downarrow_{\bar{\alpha}} \llbracket !(\text{IP} \rightarrow F[A]) \rrbracket$$

$$\downarrow_{\bar{\alpha}} \eta_{\text{obx}}$$

$$F(\llbracket !\text{Chan}(p, A) \rrbracket)$$

$$h = \llbracket P \rrbracket$$

$$\downarrow_{\bar{\alpha}} (\llbracket \varepsilon(w_p) \rrbracket \circ \sigma_p^{-1})$$

$$F(\llbracket \text{qubit} \rrbracket^{\otimes |w_p|})$$

$$\downarrow_{\bar{\alpha}} \mu \circ F(f_{\text{bind}})$$

$$F[A]$$

$$(\llbracket \varepsilon(w_p) \rrbracket = \eta_{\llbracket W_p : \text{qubit} \rrbracket})$$

$$\llbracket \vdash V : p \rightarrow A \rrbracket$$

$$= \downarrow_{\bar{\alpha}} \frac{f_v}{F(\varepsilon)} F(\llbracket ! (p \rightarrow A) \rrbracket)$$

$$\downarrow_{\bar{\alpha}} F(p \rightarrow A)$$

$$f_{\text{bind}} = \llbracket W_p : \text{qubit} \vdash V_p : A \rrbracket$$

$$= \llbracket \text{qubit} \rrbracket^{\otimes |w_p|}$$

$$\frac{\vdash V : ! (p \rightarrow A)}{\vdash V : p \rightarrow A} \quad \frac{w_p : \text{qubit} \vdash p : p}{w_p : \text{qubit} \vdash V_p : A}$$

$$\downarrow_{\bar{\alpha}} (\text{id}_{\text{obx}} \times \text{dup} \otimes \text{id}) \text{id}$$

$$\text{I} \otimes \text{I} \otimes \llbracket \text{qubit} \rrbracket^{\otimes |w_p|}$$

$$\downarrow_{\bar{\alpha}} (F(\varepsilon) \circ f_v) \otimes (\eta \circ \text{del} \otimes \sigma_p)$$

$$F(p \rightarrow A) \otimes F(p)$$

$$\downarrow_{\bar{\alpha}} \psi$$

$$F(\llbracket ! (p \rightarrow A) \rrbracket \otimes \llbracket P \rrbracket)$$

$$\downarrow_{\bar{\alpha}} F(\varepsilon_{\text{IP}}(F[A]))$$

$$F[A]$$

$$\downarrow_{\bar{\alpha}} \mu$$

$$F[A]$$

$$\llbracket W_p : \text{qubit} \vdash p : p \rrbracket$$

$$= \llbracket \text{qubit} \rrbracket^{\otimes |w_p|}$$

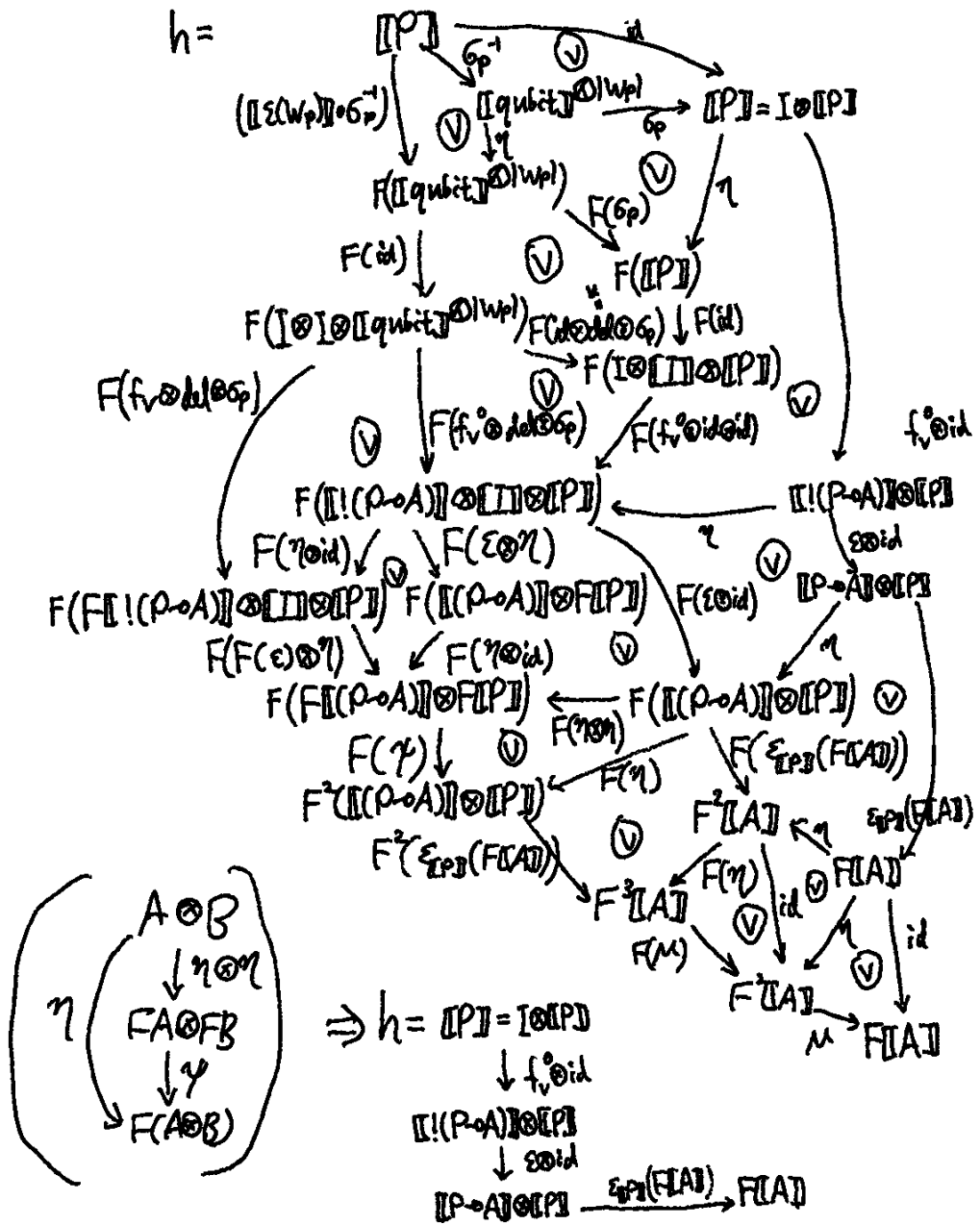
$$\downarrow_{\bar{\alpha}} \text{del} \otimes \sigma_p$$

$$\llbracket P \rrbracket$$

$$\downarrow_{\bar{\alpha}} \eta$$

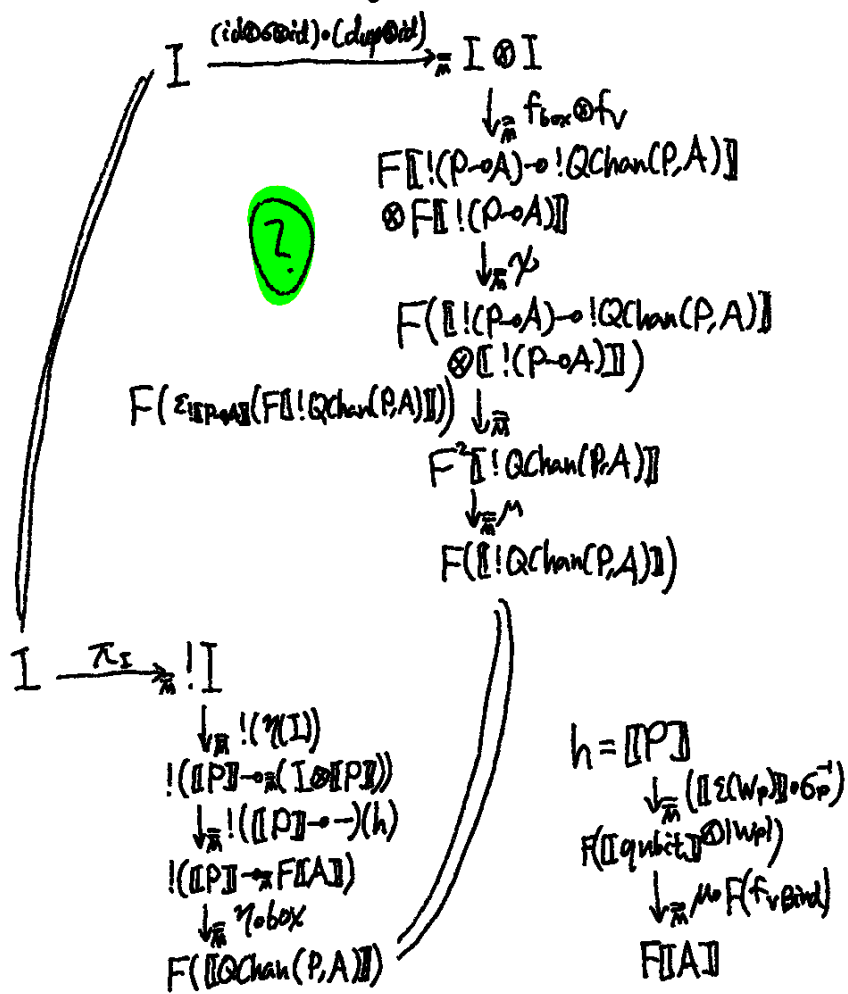
$$(\sigma_p \circ \sigma_p^{-1} = \text{id}) F(\llbracket P \rrbracket)$$

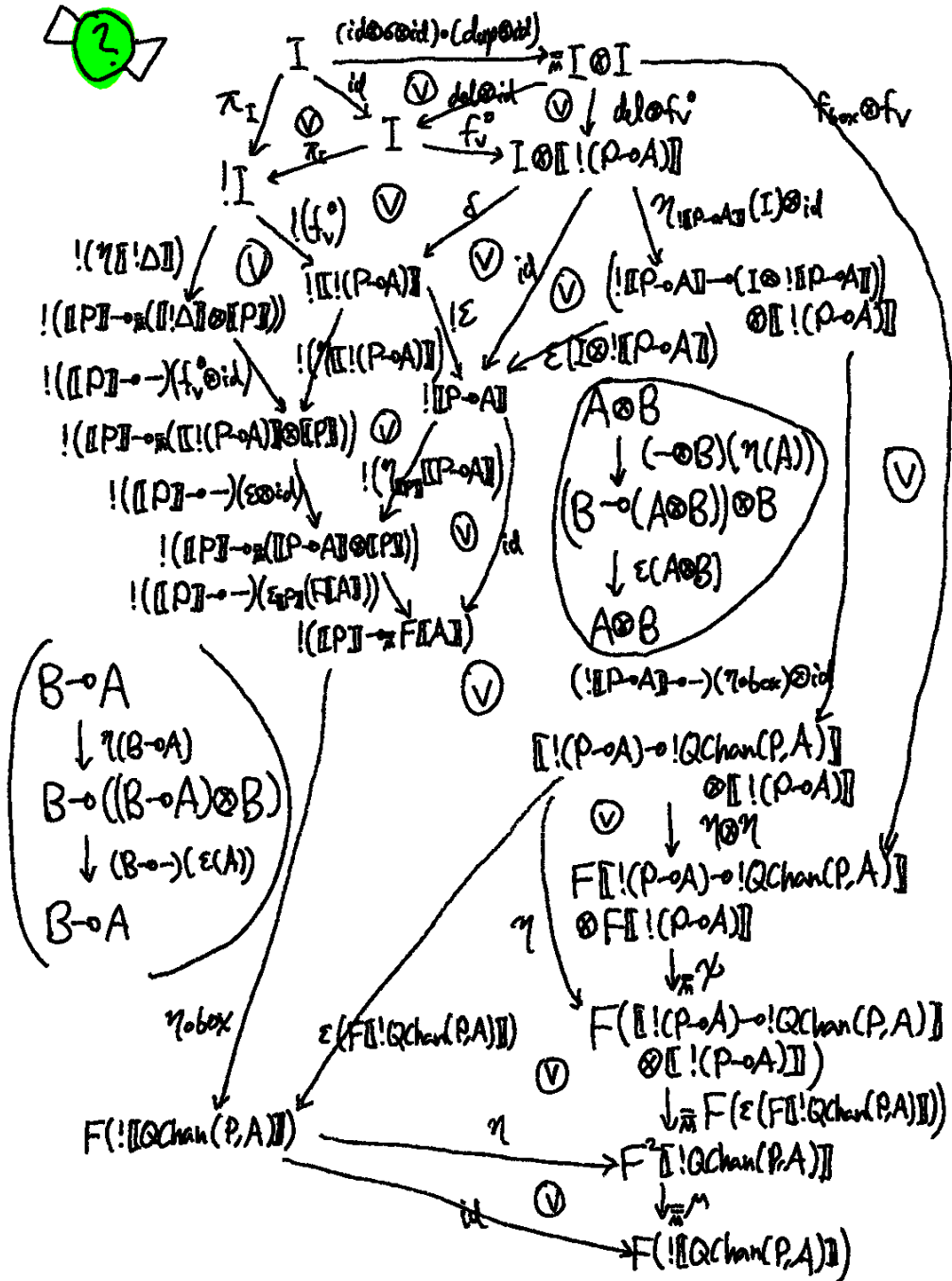
$$\sigma_p : \llbracket W_p : \text{qubit} \rrbracket \rightarrow_{\bar{\alpha}} \llbracket P \rrbracket \quad \sigma_p^{-1} : \llbracket P \rrbracket \rightarrow_{\bar{\alpha}} \llbracket W_p : \text{qubit} \rrbracket$$



$f \stackrel{?}{=} g$

The following diagram commutes:





A.5 . Unbox

$$\frac{\text{shape}(p) = \text{shape}(V) \quad \sigma = \text{bind}(p, V)}{(\varepsilon(F(V)), (\text{unbox}(p, Q, u))V) \rightarrow (\sigma(Q), \sigma(u))}$$

$$\begin{aligned} \llbracket !\Delta \vdash (\varepsilon(w), (\text{unbox}(p, Q, u))V) : A \rrbracket &= \llbracket (\llbracket \varepsilon(w) \rrbracket, f) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket w : \text{qubit} \rrbracket \xrightarrow{(\llbracket !\Delta \rrbracket \otimes -)(\llbracket \varepsilon(w) \rrbracket)} \llbracket !\Delta \rrbracket \otimes F \llbracket w : \text{qubit} \rrbracket \\ &\quad \downarrow \pi^* \\ &\quad F(\llbracket !\Delta \rrbracket \otimes \llbracket w : \text{qubit} \rrbracket) \\ &\quad \downarrow \pi \\ &\quad F(f) \\ &\quad \downarrow \pi \\ &\quad F[A] \\ &\quad \downarrow \pi^M \\ &\quad F[A] \end{aligned}$$

$w = F(V)$
 $f = \llbracket v \text{Bind}(!\Delta, w, (\text{unbox}(p, Q, u))V : A) \rrbracket$
 $= \llbracket !\Delta, w : \text{qubit} \vdash (\text{unbox}(p, Q, u))V : A \rrbracket$

$$\begin{aligned} \llbracket !\Delta \vdash (\sigma(Q), \sigma(u)) : A \rrbracket &= \llbracket (\llbracket \sigma(Q) \rrbracket, g) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q') \rrbracket \xrightarrow{(\llbracket !\Delta \rrbracket \otimes -)(\llbracket \text{in}(Q') \rrbracket)} \llbracket !\Delta \rrbracket \otimes F \llbracket \text{out}(Q') \rrbracket \\ &\quad \downarrow \pi^* \\ &\quad F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q') \rrbracket) \\ &\quad \downarrow \pi \\ &\quad F(g) \\ &\quad \downarrow \pi \\ &\quad F[A] \\ &\quad \downarrow \pi^M \\ &\quad F[A] \end{aligned}$$

$g = \llbracket v \text{Bind}(!\Delta, \text{out}(Q'), \sigma(w, A)) \rrbracket$
 where $Q' = \sigma(Q)$

$$f = [! \Delta, w: \text{qubit} \vdash (\text{unbox}(p, Q, u)) V: A]$$

$$\frac{\frac{\frac{p \vdash P \quad v \text{Bind}(! \Delta, \text{out}(Q), u, A)}{! \Delta \vdash (p, Q, u): ! \text{QChan}(P, A)}}{! \Delta \vdash (p, Q, u): \text{QChan}(P, A)}}{! \Delta \vdash \text{unbox}(p, Q, u): P \multimap A} \quad ! \Delta, w: \text{qubit} \vdash V: P}{! \Delta, w: \text{qubit} \vdash (\text{unbox}(p, Q, u)) V: A}$$

$$\begin{aligned} f &= [! \Delta] \otimes [W] \\ &\downarrow_{\pi} (id \otimes id) \circ (dup \circ id) \\ [! \Delta] \otimes [! \Delta] \otimes [W] \\ &\downarrow_{\pi} \gamma \circ (h \otimes h^2) \\ F([P \multimap A] \otimes [P]) \\ &\downarrow_{\pi} \mu \circ F(\epsilon_{PA}) \\ F[A] \end{aligned}$$

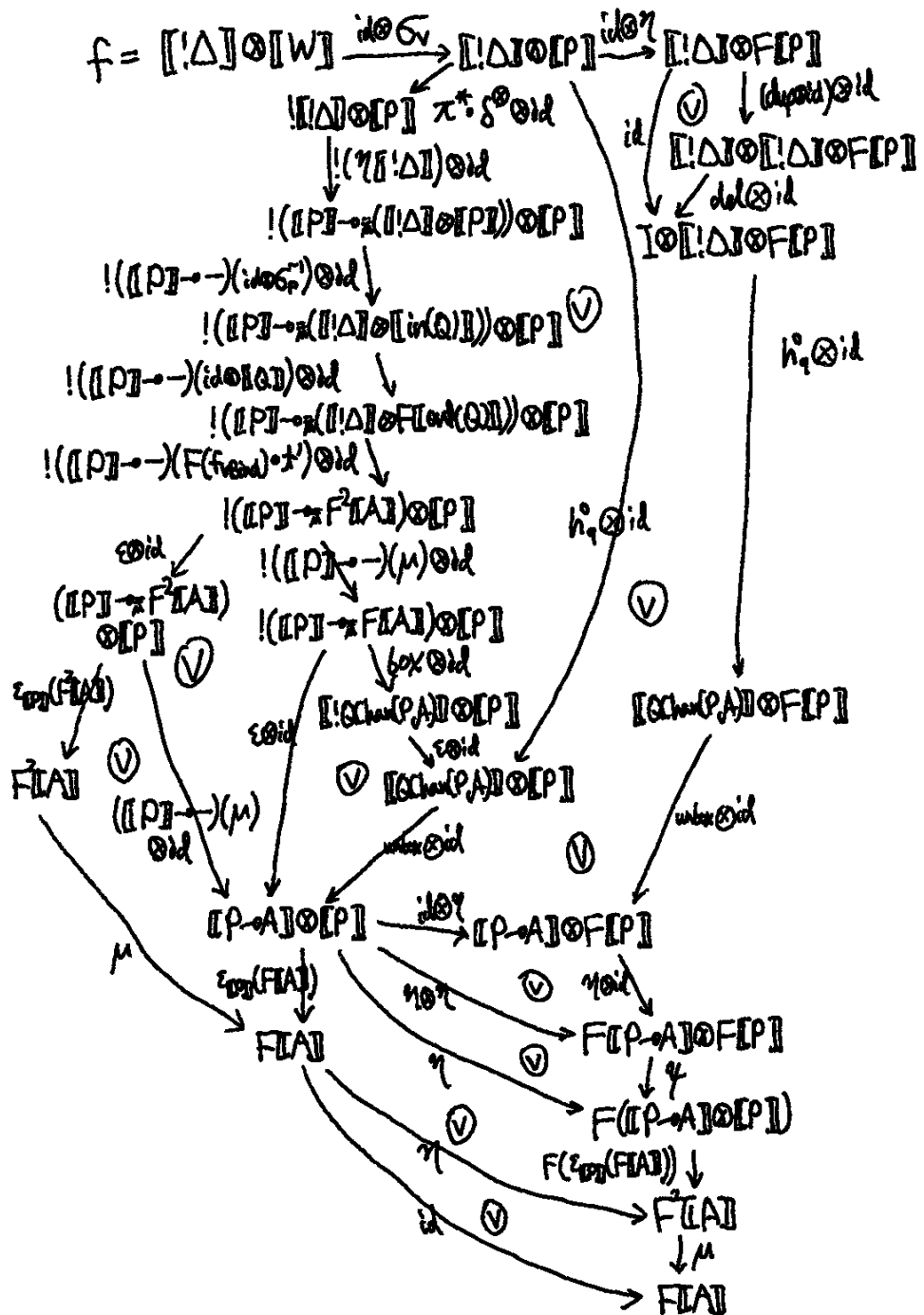
$$\begin{aligned} h^2 &= [! \Delta, w: \text{qubit} \vdash V: P] \\ &= \eta \circ \sigma_V \circ (del_{[W]} \otimes id) \end{aligned}$$

$$\begin{aligned} h^1 &= [! \Delta] \\ &\downarrow_{\pi} (id \otimes id) \circ (dup \circ id) \\ [! \Delta] \otimes [! \Delta] &\quad F([P \multimap A]) \\ &\downarrow_{\pi} \gamma \circ (h_{\text{unbox}} \otimes h_q) \quad F^2([P \multimap A]) \\ &\downarrow_{\pi} F(\epsilon_{\text{unbox}(P, A)} \circ F(\epsilon_{P \multimap A})) \uparrow_{\pi} \\ F([\text{QChan}(P, A) \multimap (P, A)] \otimes [\text{QChan}(P, A)]) &\quad (\sigma_V: [W] \rightarrow \pi [P], \\ &\quad \sigma_P: [\text{in}(Q)] \rightarrow \pi [P]) \end{aligned}$$

$$\begin{aligned} h_c &= [! \Delta] \otimes [P] \xrightarrow{id \circ \sigma_P^{-1}} [! \Delta] \otimes [\text{in}(Q)] \\ &\downarrow_{\pi} id \otimes [Q] \\ [! \Delta] \otimes F[\text{out}(Q)] &\quad \downarrow_{\pi} F(\text{fresh}) \circ \pi' \\ F[A] &\quad \downarrow_{\pi} \mu \\ F[A] \end{aligned}$$

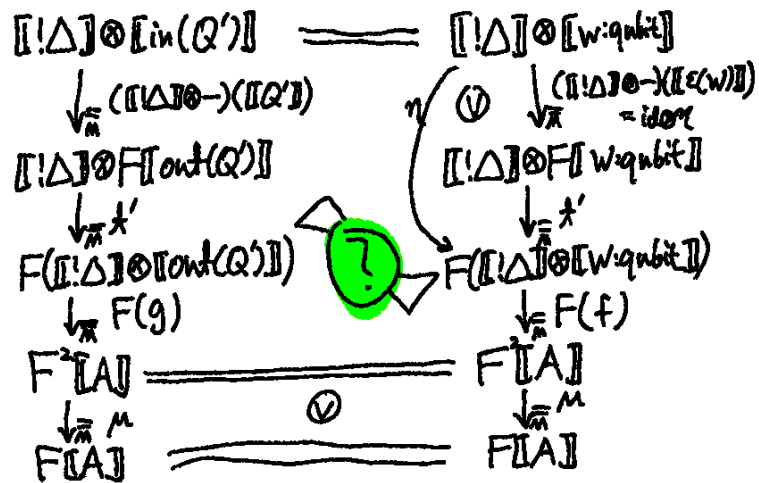
$$\begin{aligned} h_q &= [! \Delta] \\ &\downarrow_{\pi} \pi^* \circ \delta^{\otimes} \\ [! \Delta] \\ &\downarrow_{\pi} !(\eta [! \Delta]) \\ !([P] \rightarrow \pi [! \Delta] \otimes [P]) \\ &\downarrow_{\pi} !([P] \rightarrow -) (h_c) \\ !([P] \rightarrow \pi F[A]) \xrightarrow{\epsilon_{\text{box}}} [! \text{QChan}(P, A)] \\ &\downarrow_{\pi} \eta_{\text{box}} \quad \downarrow_{\pi} \eta \\ F([! \text{QChan}(P, A)]) \xrightarrow{F(\epsilon)} F[\text{QChan}(P, A)] \end{aligned}$$

$$\begin{aligned} h_{\text{unbox}} &= [! \Delta] \xrightarrow{del} I \\ &\downarrow_{\pi} \eta_{\text{unbox}} \circ \sigma_{[W]}(I) \\ [! \text{QChan}(P, A)] \multimap (I \otimes [! \text{QChan}(P, A)]) \\ &\downarrow_{\pi} ([! \text{QChan}(P, A)] \multimap \chi(\eta_{\text{unbox}})) \\ [! \text{QChan}(P, A)] \multimap F([P \multimap A]) \\ &\downarrow_{\pi} \eta \\ F([\text{QChan}(P, A) \multimap (P, A)]) \end{aligned}$$



We show the following diagram commutes:

$$([\![\epsilon(w)]\!] , f) \stackrel{?}{=} ([\![\sigma(Q)]\!] , g) \quad \left(\begin{array}{l} W = FV(W) \\ \sigma = \text{bind}(p, V) \\ Q' = \sigma(Q) \end{array} \right)$$



Note that

$$\text{in}(Q') = \text{in}(\sigma(Q)) = \sigma(\text{in}(Q)) = \sigma(p) = W$$



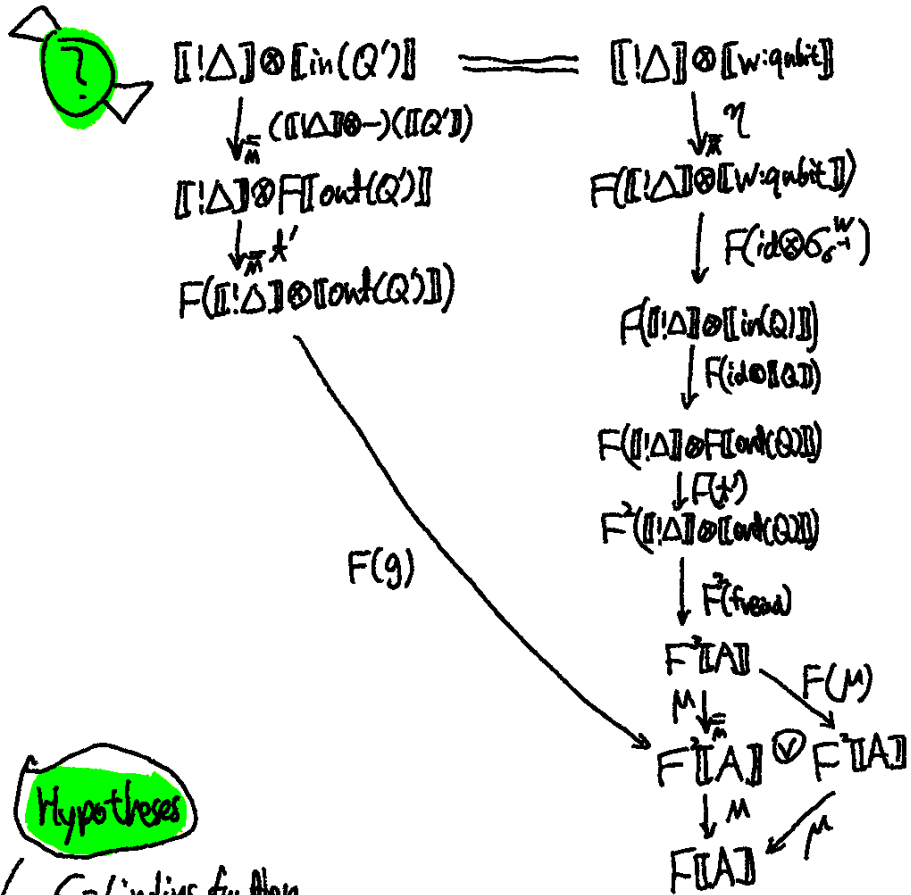
$$\begin{array}{ccc}
 [! \Delta] \otimes [in(Q')] & \cong & [! \Delta] \otimes [w:qubit] \\
 \downarrow \sqrt{\pi} ([! \Delta] \otimes \rightarrow) ([Q']) & & \downarrow \eta \\
 [! \Delta] \otimes F[out(Q')] & & F([! \Delta] \otimes [w:qubit]) \\
 \downarrow \sqrt{\pi} \sigma' & & \downarrow \sqrt{\pi} F(f) \\
 F([! \Delta] \otimes [out(Q')]) & \xrightarrow{F(g)} & F[A] \\
 & & \downarrow \mu \\
 & & F[A]
 \end{array}$$

$$f_{vBind} = [vBind(!\Delta, out(Q), u, A)]$$

$$f = [! \Delta] \otimes [w]$$

$$[! \Delta] \otimes [in(Q)] \xrightarrow{id \otimes \sigma'} [! \Delta] \otimes F[out(Q)] \xrightarrow{F(f_{vBind}) \circ \sigma'} F[A] \xrightarrow{\mu} F[A]$$

$$g = [vBind(!\Delta, out(Q'), \sigma(w), A)]$$



Hypotheses

- $\sigma = \text{binding function}$
- $Q' = \sigma(Q)$
- $g = [[vBind(!Δ, out(Q'), \sigma(u), A)]]$
- $f_{vBind} = [[vBind(!Δ, out(Q), u, A)]]$
- $w = in(Q')$

We prove  by induction on Q :

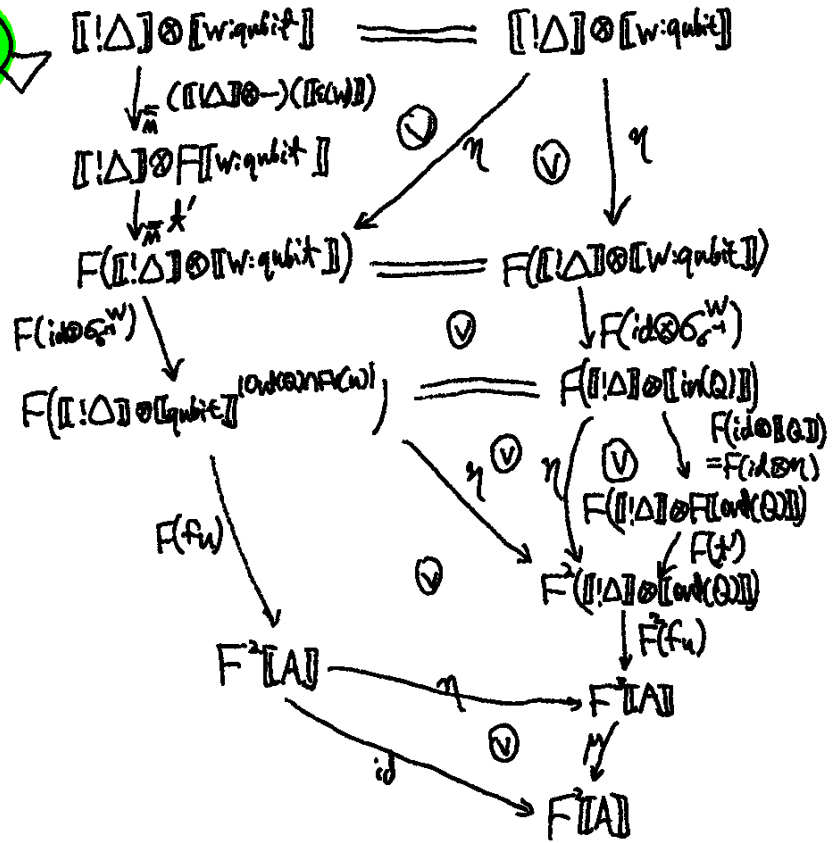
$$\textcircled{1} Q = \varepsilon(X) : X = \text{supp}(p), \quad \sigma(X) = \sigma(\text{supp}(p)) = FV(p) = W$$

and $Q' = \varepsilon(\sigma(X)) = \varepsilon(W)$

$$f_{\text{vBind}} = \llbracket \text{vBind}(!\Delta, X, u, A) \rrbracket = \llbracket !\Delta \rrbracket \otimes \llbracket X \rrbracket \xrightarrow{f_u} \llbracket A \rrbracket$$

$$f_u = \llbracket !\Delta, (X \cap FV(u)) : \text{qubit} \vdash u : A \rrbracket$$

$$\begin{aligned} \mathcal{G} &= \llbracket \text{vBind}(!\Delta, \text{out}(Q'), \sigma(u), A) \rrbracket \\ &= \llbracket !\Delta, w : \text{qubit} \vdash \sigma(u) : A \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket \text{qubit} \rrbracket \xrightarrow{\text{id} \otimes \sigma^w} \llbracket !\Delta \rrbracket \otimes \llbracket \text{qubit} \rrbracket^{\{\text{out}(Q')\}} \\ &\quad \downarrow f_u \\ &\quad \llbracket A \rrbracket \end{aligned}$$



$$\begin{aligned}
 Q &= \varepsilon(X) \\
 Q' &= \varepsilon(w) = \sigma(Q)
 \end{aligned}$$

$$\textcircled{2} Q^U = U(Y) Q: \llbracket Q^U \rrbracket = \llbracket Q \rrbracket \circ \llbracket U(Y) \rrbracket^\circ$$

$$\text{in}(Q^U) = \text{supp}(p) = \text{in}(Q), \text{out}(Q^U) = \text{out}(Q)$$

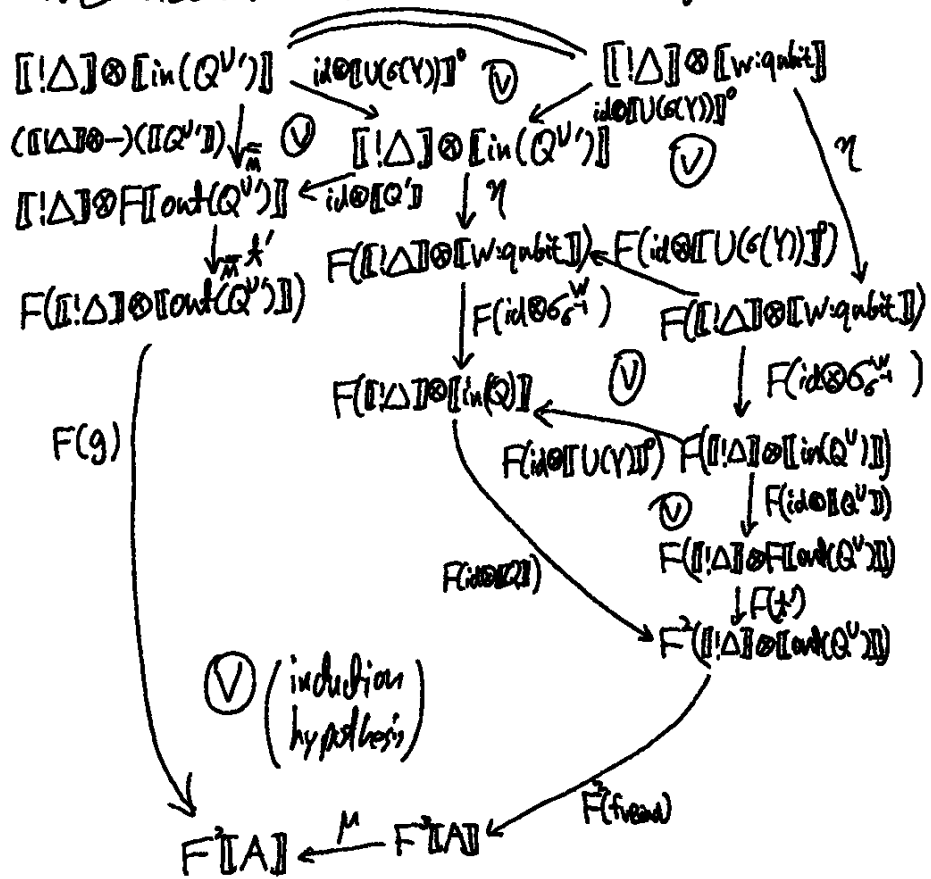
$$\text{and } Q' = \sigma(Q) \quad Q^{U'} = \sigma(Q^U) = U(\sigma(Y)) Q' \quad (W = FV(U) = \sigma(\text{in}(Q)))$$

From the induction hypothesis:

$$\begin{array}{ccc} \llbracket \Delta \rrbracket \otimes \llbracket \text{in}(Q') \rrbracket & \equiv & \llbracket \Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket \\ \downarrow_{\bar{m} (\llbracket \Delta \rrbracket \otimes \rightarrow) (\llbracket Q' \rrbracket)} & & \downarrow_{\bar{m} ?} \\ \llbracket \Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket] & & F[\llbracket \Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket] \\ \downarrow_{\bar{m} \lambda'} & & \downarrow_{F(\text{id} \otimes \sigma_s^{w})} \\ F[\llbracket \Delta \rrbracket \otimes \llbracket \text{out}(Q') \rrbracket] & & F[\llbracket \Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket] \\ & & \downarrow_{F(\text{id} \otimes \text{id})} \\ & & F[\llbracket \Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket]] \\ & & \downarrow_{F(\lambda')} \\ & & F^2[\llbracket \Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket] \\ & & \downarrow_{F^2(\text{fresh})} \\ & & F^2[\Delta] \\ & & \downarrow_{\bar{m} M} \\ & & F^2[\Delta] \end{array}$$

$F(g)$

We need to show the following:



$$Q^{u'} = U(\sigma(\gamma)) Q' \quad [[Q^{u'}]] = [[Q']] \circ [[U(\sigma(\gamma))]^\circ]$$

$$g^{u'} = [[v\text{Bind}(\Delta, \overset{\text{out}(Q)}{\text{out}(Q^{u'})}, \sigma(u), A)]] = g$$

$$f_{v\text{Bind}}^u = [[v\text{Bind}(\Delta, \overset{\text{out}(Q)}{\text{out}(Q^u)}, u, A)]] = f_{v\text{Bind}}$$

$$\textcircled{3} Q^f = \text{free} \vee Q \quad \llbracket Q^f \rrbracket = \llbracket Q \rrbracket \circ \llbracket \text{free}(v) \rrbracket$$

$$\text{in}(Q^f) = \text{supp}(p) = \text{in}(Q) \cup \{v\}, \quad \text{out}(Q^f) = \text{out}(Q)$$

$$w^f = FV(v) = \text{in}(\sigma(Q^f))$$

$$\text{and } Q' = \sigma(Q)$$

$$Q^{f'} = \sigma(Q^f) = \text{free } \sigma(v) \ Q'$$

From the induction hypothesis:

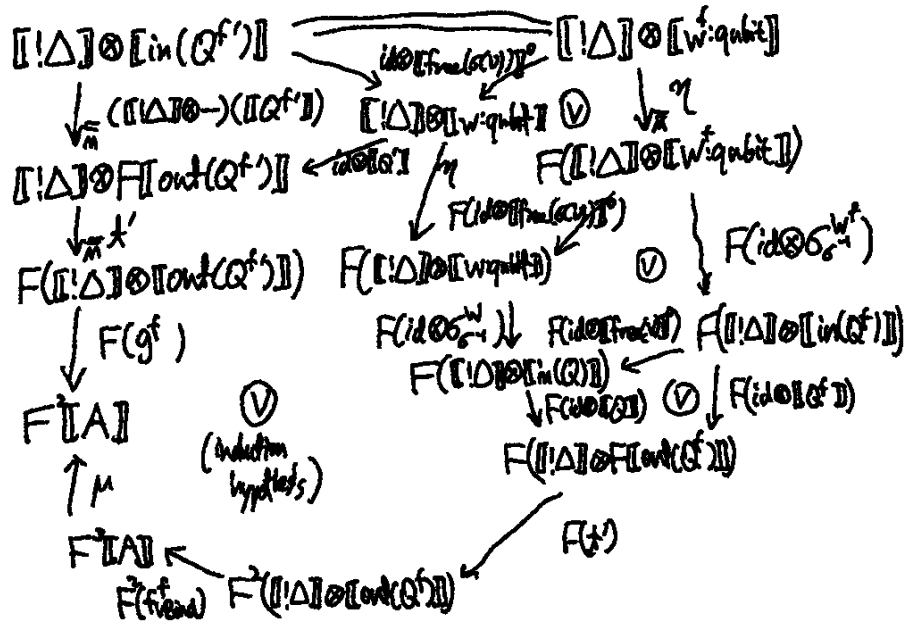
$$\begin{array}{ccc}
 \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q') \rrbracket & \equiv & \llbracket !\Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket \\
 \downarrow \pi_{\llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q') \rrbracket} & & \downarrow \pi_{\llbracket !\Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket} \\
 \llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket] & & F[\llbracket !\Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket] \\
 \downarrow \pi_{\llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket]} & & \downarrow F(\text{id} \otimes \sigma_w) \\
 F[\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket] & & F[\llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket] \\
 \downarrow F(g') & & \downarrow F(\text{id} \otimes \sigma) \\
 F[A] & & F[\llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket]] \\
 & \swarrow \text{M} & \swarrow F(\sigma) \\
 & F[A] & F[\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket] \\
 & \swarrow F(\text{free}) & \\
 & F[A] &
 \end{array}$$

where $w = \text{in}(Q')$, σ is a binding function

$$g' = \llbracket v\text{Bind}(!\Delta, \text{out}(Q'), \sigma(v), A) \rrbracket$$

$$f_v\text{Bind} = \llbracket v\text{Bind}(!\Delta, \text{out}(Q), u, A) \rrbracket$$

We need to show



where σ is binding function

$$\begin{aligned}
 Q_a^f &= \sigma(\text{in}(Q^f)) = \sigma(\text{in}(Q)) \cup \{\sigma(v)\} = Q_a \cup \{\sigma(v)\} \\
 W^f &= \text{in}(Q^f) = \text{in}(Q) \cup \{\sigma(v)\} = W \cup \{\sigma(v)\} \\
 g^f &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q^f), \sigma(u), A) \rrbracket \\
 &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q), \sigma(u), A) \rrbracket = g \\
 f_{v \text{Bind}}^f &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q^f), u, A) \rrbracket \\
 &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q), u, A) \rrbracket = f_{v \text{Bind}}
 \end{aligned}$$

$$\textcircled{4} Q^i = \text{init } b \vee Q \quad \llbracket Q^i \rrbracket = \llbracket Q \rrbracket \circ \llbracket \text{init}(b, v) \rrbracket$$

$$\text{in}(Q^i) = \text{supp}(p) = \text{in}(Q) \setminus \{v\}, \quad \text{out}(Q^i) = \text{out}(Q)$$

$$\text{and } Q' = \sigma(Q)$$

$$Q^{i'} = \sigma(Q^i) = \text{init } b \sigma(v) Q'$$

From the induction hypothesis:

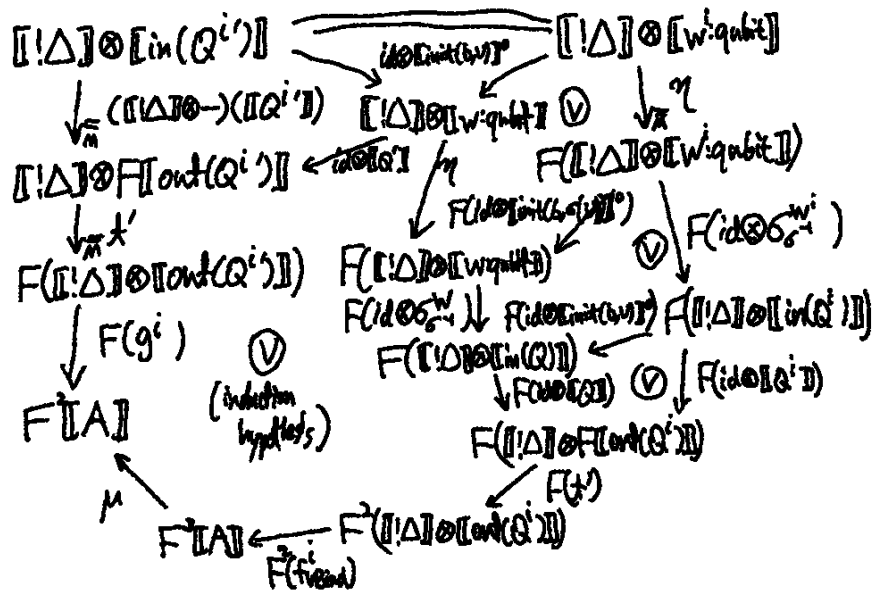
$$\begin{array}{ccc}
 \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q^i) \rrbracket & \equiv & \llbracket !\Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket \\
 \downarrow \pi_{\text{in}} (\llbracket !\Delta \rrbracket \otimes -) (\llbracket Q^i \rrbracket) & & \downarrow \eta \\
 \llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket] & & F[\llbracket !\Delta \rrbracket \otimes \llbracket w:\text{qubit} \rrbracket] \\
 \downarrow \pi_{\text{out}}^* & & \downarrow F(\text{id} \otimes \sigma_w^{-1}) \\
 F[\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q^i) \rrbracket] & & F[\llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket] \\
 \downarrow F(g) & & \downarrow F(\text{id} \otimes \text{id}) \\
 F[A] & & F[\llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket]] \\
 & \swarrow \mu & \swarrow F(\sigma) \\
 & F[A] & F[\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket] \\
 & \nearrow F(\text{fresh}) & \\
 & F[A] &
 \end{array}$$

where $w = \text{in}(Q^i)$, σ is a binding function

$$g = \llbracket v\text{Bind}(!\Delta, \text{out}(Q^i), \sigma(w), A) \rrbracket$$

$$f_v\text{Bind} = \llbracket v\text{Bind}(!\Delta, \text{out}(Q), u, A) \rrbracket$$

We need to show



where

σ is binding function

$w^i = \text{in}(Q^{i'}) = \text{in}(Q^i) \setminus \{\sigma(v)\} = w \setminus \{\sigma(v)\}$

$g^i = \llbracket \text{vBind}(!\Delta, \text{out}(Q^{i'}), \sigma(w), A) \rrbracket$
 $= \llbracket \text{vBind}(!\Delta, \text{out}(Q^i), \sigma(w), A) \rrbracket = g$

$f_{\text{vBind}}^i = \llbracket \text{vBind}(!\Delta, \text{out}(Q^i), u, A) \rrbracket$
 $= \llbracket \text{vBind}(!\Delta, \text{out}(Q), u, A) \rrbracket = f_{\text{vBind}}$

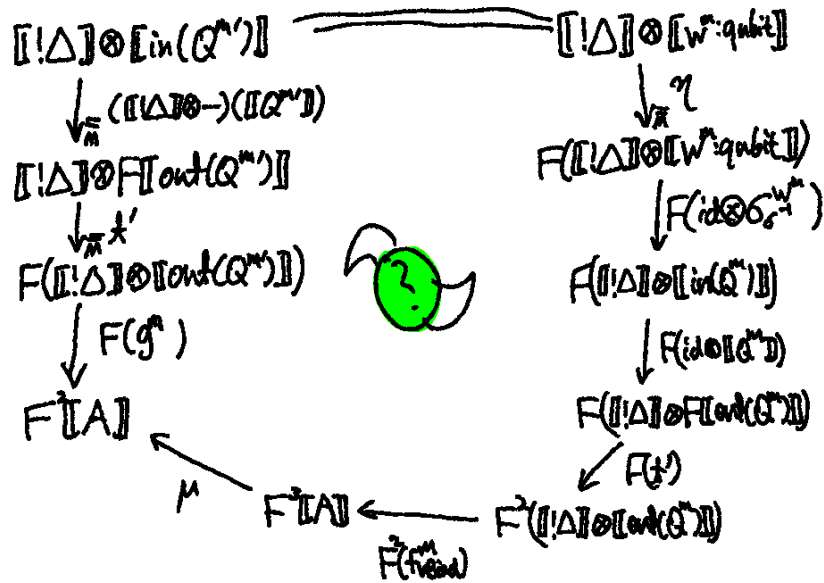
$$\begin{aligned}
 (5) \quad Q^m &= \text{meas} \vee Q_1, Q_2 \quad \llbracket Q^m \rrbracket = \llbracket \text{in}(\text{meas} \vee Q_1, Q_2) \rrbracket \\
 \text{in}(Q^m) &= \text{in}(Q_{1|2}), \text{out}(Q^m) = [\text{out}(Q_1), \text{out}(Q_2)] \quad \begin{array}{l} \downarrow \text{bit} \\ F(\llbracket \text{in}(\text{meas} \vee Q_1, Q_2) \rrbracket \\ + \llbracket \text{in}(\text{meas} \vee Q_1, Q_2) \rrbracket) \\ \downarrow F(\llbracket Q_1 \rrbracket \circ \llbracket \text{meas}(u_1) \rrbracket \\ + \llbracket Q_2 \rrbracket \circ \llbracket \text{meas}(u_2) \rrbracket) \\ F(F(\llbracket \text{out}(Q_1) \rrbracket) + F(\llbracket \text{out}(Q_2) \rrbracket)) \\ \downarrow F(\text{merge}) \\ F^2(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \\ \downarrow M \\ F(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \end{array} \\
 u &= [u_1, u_2] \\
 \text{and } Q_{1|2}' &= \sigma(Q_{12}) \\
 Q^{m'} &= \sigma(Q^m) = \text{meas} \circ (v) \quad Q_1' \quad Q_2'
 \end{aligned}$$

From the induction hypothesis:

$$\begin{array}{ccc}
 \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q_{12}') \rrbracket & \equiv & \llbracket !\Delta \rrbracket \otimes \llbracket w : \text{qubit} \rrbracket \\
 \downarrow \pi_{\text{in}} (\llbracket !\Delta \rrbracket \otimes -) (\llbracket Q_{12}' \rrbracket) & & \downarrow \pi \\
 \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q_{12}') \rrbracket) & & F(\llbracket !\Delta \rrbracket \otimes \llbracket w : \text{qubit} \rrbracket) \\
 \downarrow \pi \circ \pi' & & \downarrow F(\text{id} \otimes \sigma_{12}^{M'}) \\
 F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q_{12}') \rrbracket) & \xrightarrow{\square} & F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q_{12}') \rrbracket) \\
 \downarrow F(g_{12}) & & \downarrow F(\text{id} \otimes \sigma_{12}) \\
 F(!A) & \xleftarrow{\square} & F(\llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q_{12}') \rrbracket)) \\
 & & \downarrow F(\dagger) \\
 & & F^2(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q_{12}') \rrbracket) \\
 & & \downarrow F(\text{id} \otimes \sigma_{12}) \\
 & & F^2(!A)
 \end{array}$$

where $w = \text{in}(Q_{12}')$, σ is a binary function
 $g_{12} = \llbracket v \text{Bind}(!\Delta, \text{out}(Q_{12}'), \sigma(u_{12}), A) \rrbracket$
 $f_{v \text{Bind}} = \llbracket v \text{Bind}(!\Delta, \text{out}(Q_{12}'), u_{12}, A) \rrbracket$

We need to show



σ is binding function

$$w^m = \text{in}(Q^{m'}) = \text{in}(Q_{1|2}) = w$$

$$g^m = [\text{vBind}(!\Delta, \text{out}(Q^{m'}), \sigma(u), A)]$$

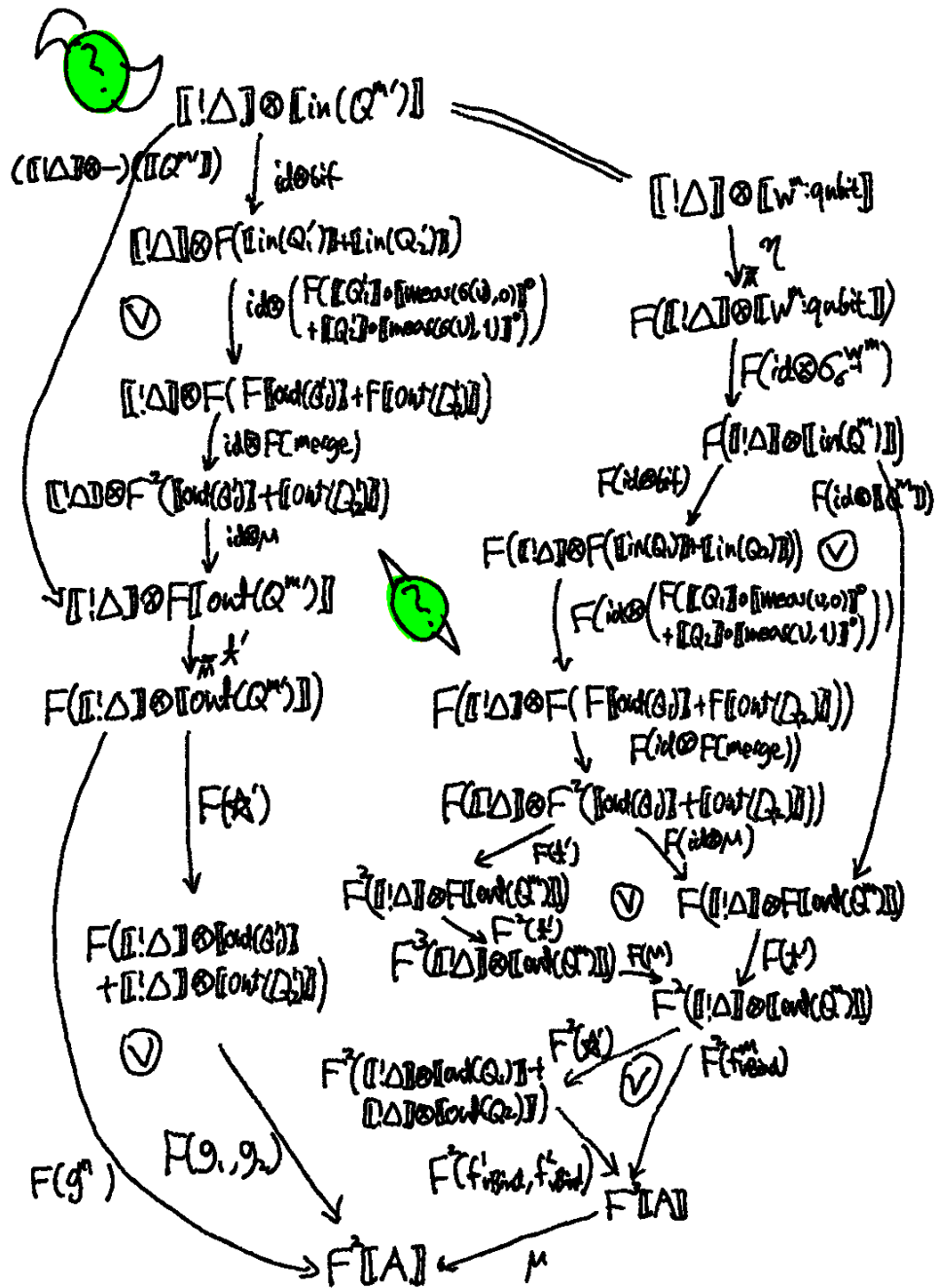
where

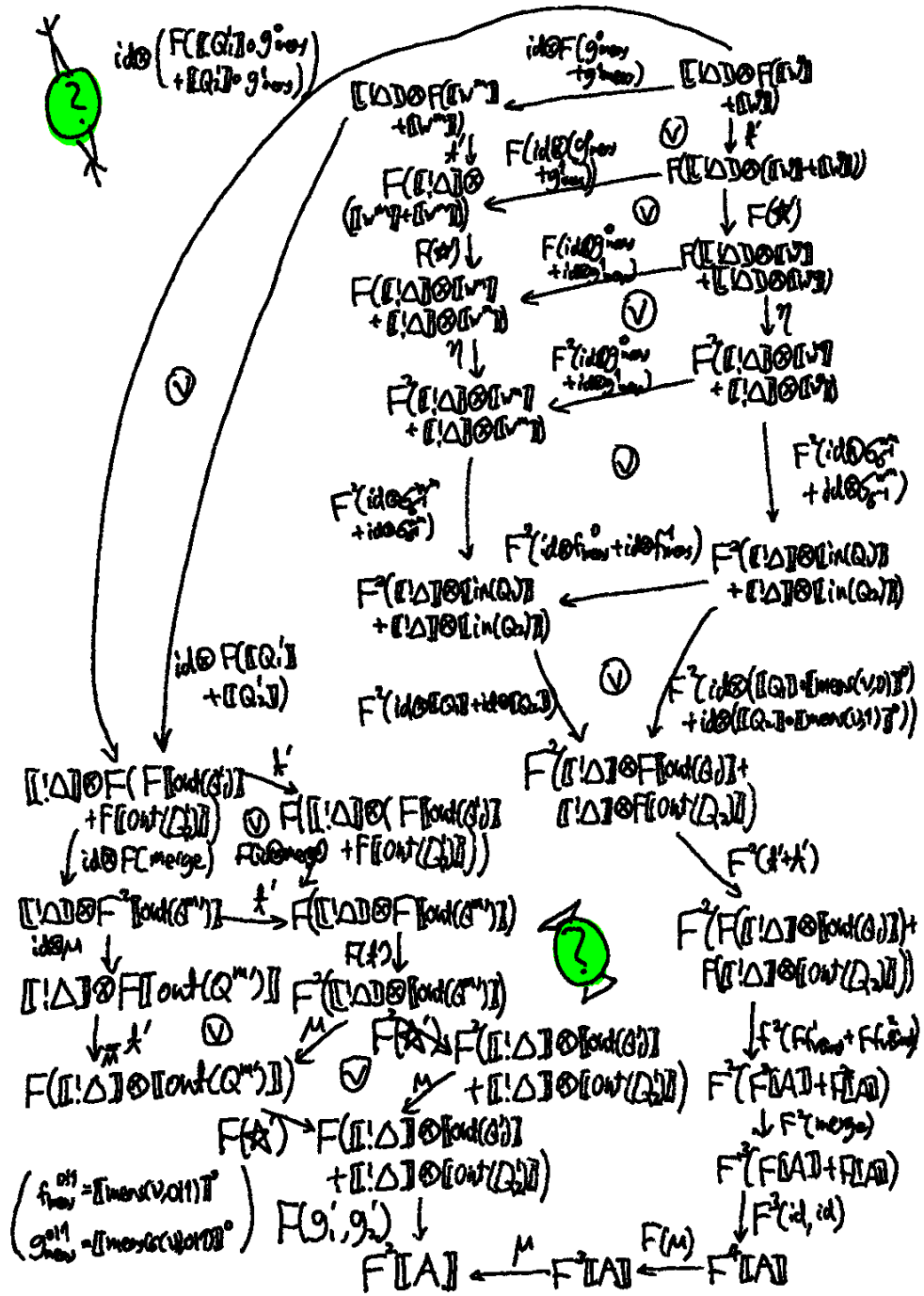
$$= [\text{vBind}(!\Delta, [\text{out}(Q_1), \text{out}(Q_2)], [\sigma(u_1), \sigma(u_2)], A)] = (g_1, g_2) \circ \star$$

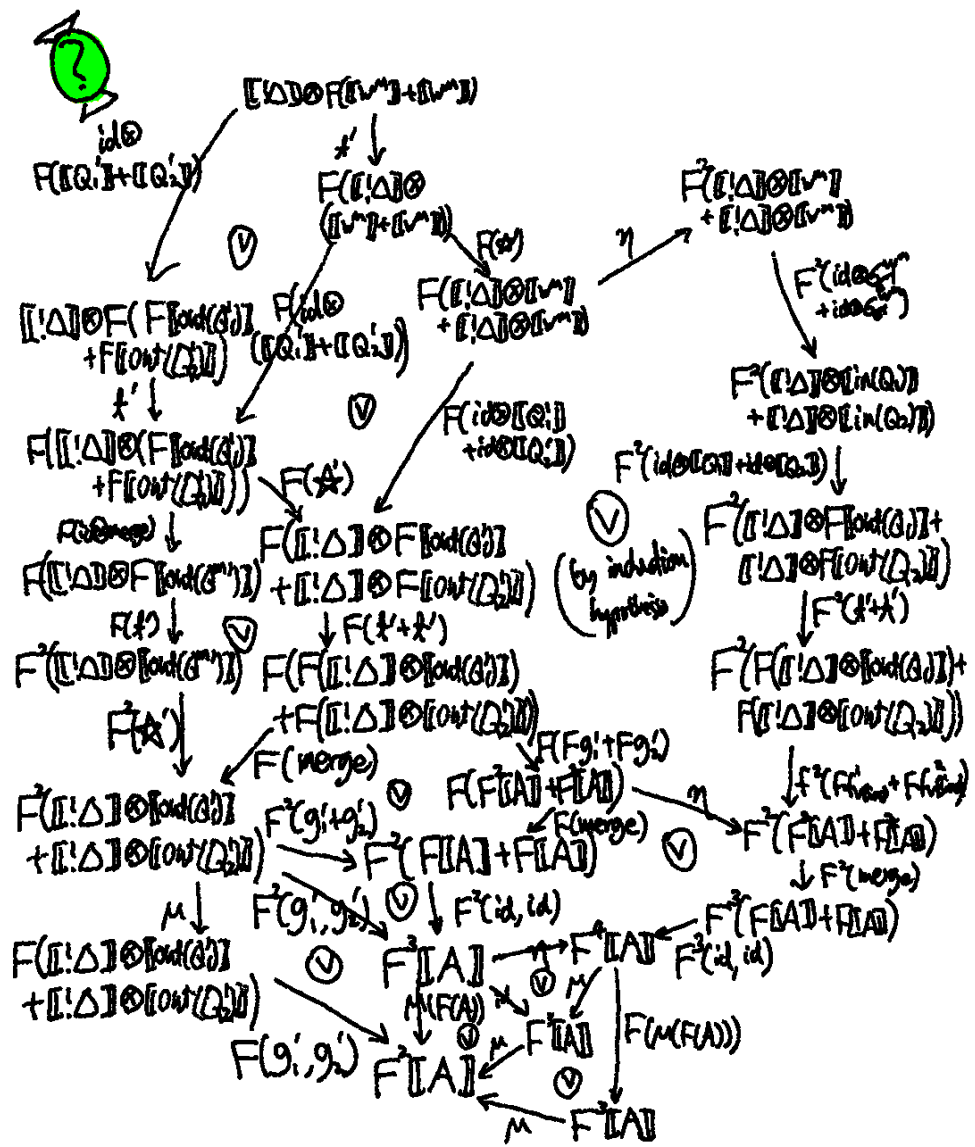
$$f_{\text{vBind}}^m = [\text{vBind}(!\Delta, \text{out}(Q^m), u, A)]$$

$$= [\text{vBind}(!\Delta, [\text{out}(Q_1), \text{out}(Q_2)], [u, u_2], A)]$$

$$= (f_{\text{vBind}}^1, f_{\text{vBind}}^2) \circ \star'$$







A.6 . Congruence: quantum channel

$$\frac{(Q, m) \rightarrow (Q', m')}{(\mathcal{E}(\phi), (p, Q, m)) \rightarrow (\mathcal{E}(\phi), (p, Q', m')) \quad (A' = !Q\text{Chan}(P, A))}$$

$$\llbracket !\Delta \vdash (\mathcal{E}(\phi), (p, Q, m)) : A' \rrbracket = \llbracket (\llbracket \mathcal{E}(\phi) \rrbracket, f) \rrbracket$$

$$\llbracket !\Delta \vdash (\mathcal{E}(\phi), (p, Q', m')) : A' \rrbracket = \llbracket (\llbracket \mathcal{E}(\phi) \rrbracket, g) \rrbracket$$

$$f = \llbracket !\Delta \vdash (p, Q, m) : A' \rrbracket \quad g = \llbracket !\Delta \vdash (p, Q', m') : A' \rrbracket$$

$$\frac{p \neq P \quad v\text{Bind}(!\Delta, \text{out}(Q), m, A)}{!\Delta \vdash (p, Q, m) : !Q\text{Chan}(P, A)} \quad \frac{p \neq P \quad v\text{Bind}(!\Delta, \text{out}(Q'), m', A)}{!\Delta \vdash (p, Q', m') : !Q\text{Chan}(P, A)}$$

$$f = \llbracket !\Delta \rrbracket \xrightarrow{\pi^* \circ \delta^*} \llbracket !\Delta \rrbracket$$

$$h_f = \llbracket !\Delta \rrbracket \otimes \llbracket P \rrbracket \xrightarrow{\downarrow_{\bar{\alpha}} \text{id} \otimes \sigma_P} \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \xrightarrow{\downarrow_{\bar{\alpha}} [\!| Q \!|]} \llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket] \xrightarrow{\downarrow_{\bar{\alpha}} \uparrow} F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \xrightarrow{\downarrow_{\bar{\alpha}} \mu \circ F(f_{v\text{Bind}})} F[A]$$

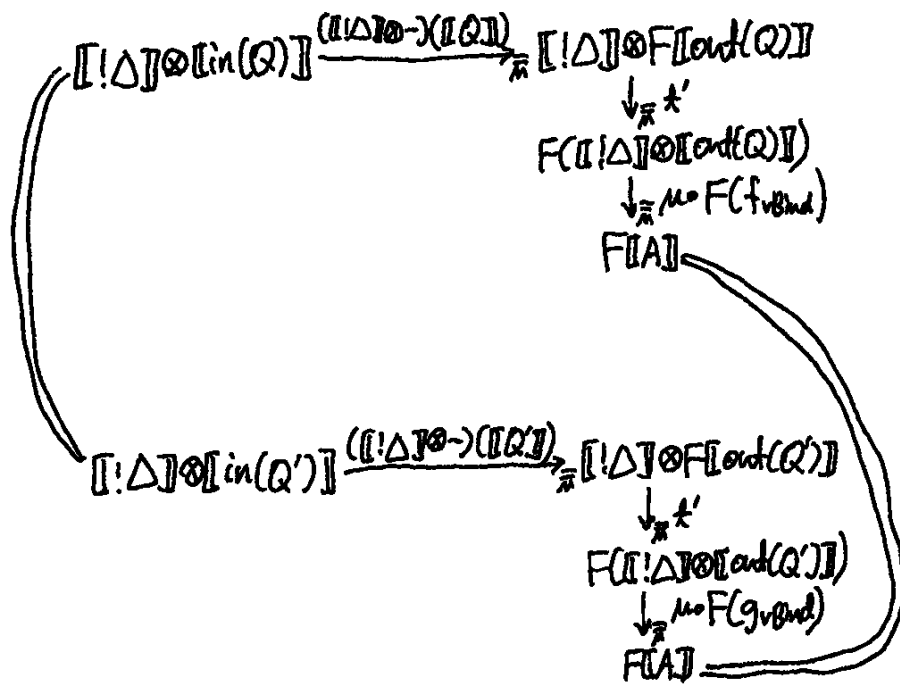
$$g = \llbracket !\Delta \rrbracket \xrightarrow{\pi^* \circ \delta^*} \llbracket !\Delta \rrbracket$$

$$h_g = \llbracket !\Delta \rrbracket \otimes \llbracket P \rrbracket \xrightarrow{\downarrow_{\bar{\alpha}} \text{id} \otimes \sigma_P} \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \xrightarrow{\downarrow_{\bar{\alpha}} [\!| Q' \!|]} \llbracket !\Delta \rrbracket \otimes F[\llbracket \text{out}(Q) \rrbracket] \xrightarrow{\downarrow_{\bar{\alpha}} \uparrow} F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \xrightarrow{\downarrow_{\bar{\alpha}} \mu \circ F(g_{v\text{Bind}})} F[A]$$

$$f_{v\text{Bind}} = \llbracket v\text{Bind}(!\Delta, \text{out}(Q), m, A) \rrbracket \quad g_{v\text{Bind}} = \llbracket v\text{Bind}(!\Delta, \text{out}(Q'), m', A) \rrbracket$$

By induction hypothesis:

$$\llbracket !\Delta \vdash (Q, m) : A \rrbracket = \llbracket !\Delta \vdash (Q', m') : A \rrbracket$$



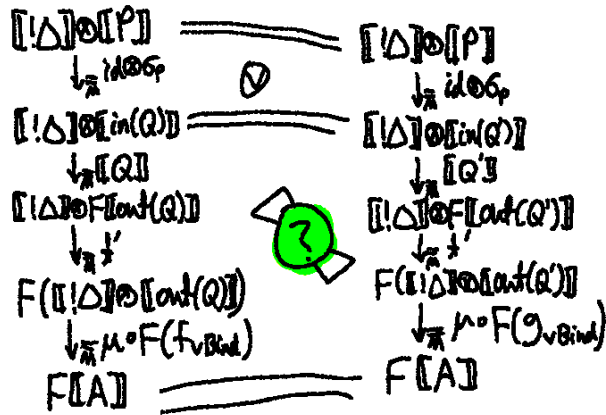
Note that


$$h_f = h_g \Rightarrow f = g \Rightarrow f' = g' \Rightarrow$$

$$\llbracket !\Delta \vdash (\varepsilon(w), (p, Q, m)) : A \rrbracket = \llbracket !\Delta \vdash (\varepsilon(w), (p, Q', m')) : A \rrbracket$$

Hence, it suffices to show that $(h_f = h_g)$:

Following diagram commutes:



 follows from the induction hypothesis.

A.7 . Congruence: on the right of application

$$\frac{(\varepsilon(W_M), M) \rightarrow (Q, m) \quad \text{all}(Q) \cap W_N = \emptyset}{(\varepsilon(W_M \cup W_N), MN) \rightarrow (\text{extend}(Q, W_N), mN)}$$

We show that
 $\llbracket !\Delta \vdash (\varepsilon(W_M \cup W_N), MN) : A \rrbracket = \llbracket !\Delta \vdash (\text{extend}(Q, W_N), mN) : A \rrbracket$
 given that $\llbracket !\Delta \vdash (\varepsilon(W_M), M) : A' \rrbracket = \llbracket !\Delta \vdash (Q, m) : A' \rrbracket$

where $A' = (A_m \rightarrow A)$.

(let $w = W_M \cup W_N$ and $Q' = \text{extend}(Q, W_N)$)

$$\begin{aligned} & \llbracket !\Delta \vdash (\varepsilon(W), MN) : A \rrbracket \\ &= \llbracket ([z(W)], f_{\text{vbind}}) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket W \rrbracket \\ & \quad \downarrow (\llbracket W \rrbracket \otimes) \cdot (\llbracket \varepsilon(W) \rrbracket) \\ & \llbracket !\Delta \rrbracket \otimes F \llbracket W \rrbracket \\ & \quad \downarrow \dagger' \\ & F(\llbracket !\Delta \rrbracket \otimes \llbracket W \rrbracket) \\ & \quad \downarrow F(f_{\text{vbind}}) \\ & F^2 \llbracket A \rrbracket \\ & \quad \downarrow \bar{M} \\ & F \llbracket A \rrbracket \end{aligned}$$

$$\begin{aligned} & \llbracket !\Delta \vdash (Q', mN) : A \rrbracket \\ &= \llbracket ([Q'], g_{\text{vbind}}) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \\ & \quad \downarrow (\llbracket \text{in}(Q) \rrbracket \otimes) \cdot (\llbracket Q \rrbracket) \\ & \llbracket !\Delta \rrbracket \otimes F \llbracket \text{out}(Q) \rrbracket \\ & \quad \downarrow \dagger' \\ & F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \\ & \quad \downarrow F(g_{\text{vbind}}) \\ & F^2 \llbracket A \rrbracket \\ & \quad \downarrow \bar{M} \\ & F \llbracket A \rrbracket \end{aligned}$$

$$f_{\text{vbind}} = \llbracket \text{vbind}(!\Delta, W_M \cup W_N, MN, A) \rrbracket$$

$$g_{\text{vbind}} = \llbracket \text{vbind}(!\Delta, \text{out}(Q'), mN, A) \rrbracket$$

From $! \Delta \vdash (\mathcal{E}(W_M), M) : A'$, we know that
 $! \Delta, W_M : \text{qubit} \vdash M : A'$

Then from $! \Delta \vdash (\mathcal{E}(W), MN) : A$, we can obtain the following derivation:

$$\frac{! \Delta, W_M : \text{qubit} \vdash M : A' \quad ! \Delta, W_N : \text{qubit} \vdash N : A_n}{! \Delta, W : \text{qubit} \vdash MN : A}$$

Let $f_N = [! \Delta, (W_N : \text{qubit}) \vdash N : A_n]$, $f_M = [! \Delta, (W_M : \text{qubit}) \vdash M : A']$.

From $\frac{W_M \wedge FV(! \Delta) = \emptyset \quad ! \Delta, (W_M : \text{qubit}) \vdash M : A'}{\nu \text{Bind}(! \Delta, W_M, M, A')}$

and

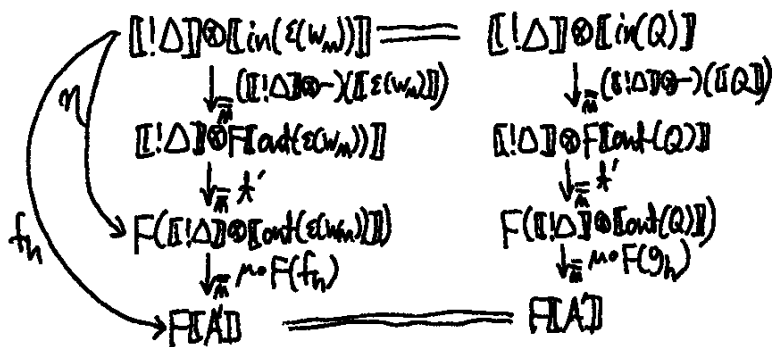
$$\frac{! \Delta, (W_M : \text{qubit}) \vdash M : A' \quad ! \Delta, (W_N : \text{qubit}) \vdash N : A_n}{W \wedge FV(! \Delta) = \emptyset \quad ! \Delta, (W : \text{qubit}) \vdash MN : A}$$

we get $f_h = [\nu \text{Bind}(! \Delta, W_M, M, A')] = [! \Delta] \otimes [W_M] \xrightarrow{f_M} F[A']$
 $f_{\nu \text{Bind}} = [\nu \text{Bind}(! \Delta, W, MN, A)] = [! \Delta] \otimes [W]$

$$\begin{array}{c} [! \Delta] \otimes [W_M] \otimes [W_N] \\ \downarrow \text{id} \otimes \sigma_W \\ [! \Delta] \otimes [W_M] \otimes [W_N] \\ \downarrow \text{dup} \otimes \text{id} \\ [! \Delta] \otimes [! \Delta] \otimes [W_M] \otimes [W_N] \\ \leftarrow \text{id} \otimes \sigma \otimes \text{id} \\ [! \Delta] \otimes [W_M] \otimes [! \Delta] \otimes [W_N] \\ \downarrow f_M \otimes f_N \\ F[A_n \circ A] \otimes F[A_n] \xrightarrow{\gamma} F([A_n \circ A] \otimes [A_n]) \\ \downarrow \text{eval} \\ F^2[A] \xrightarrow{\mu} F[A] \end{array}$$

By induction hypothesis: $(\star A' = (A_n \circ A))$

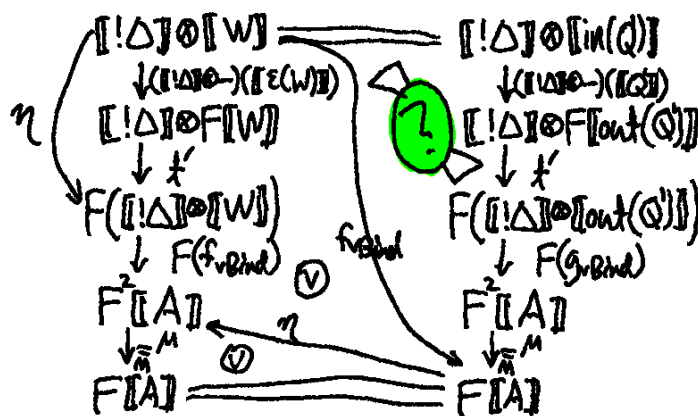
$$\llbracket !\Delta \vdash (\varepsilon w_m), M \rrbracket : A' = \llbracket !\Delta \vdash (Q, m) \rrbracket : A'$$



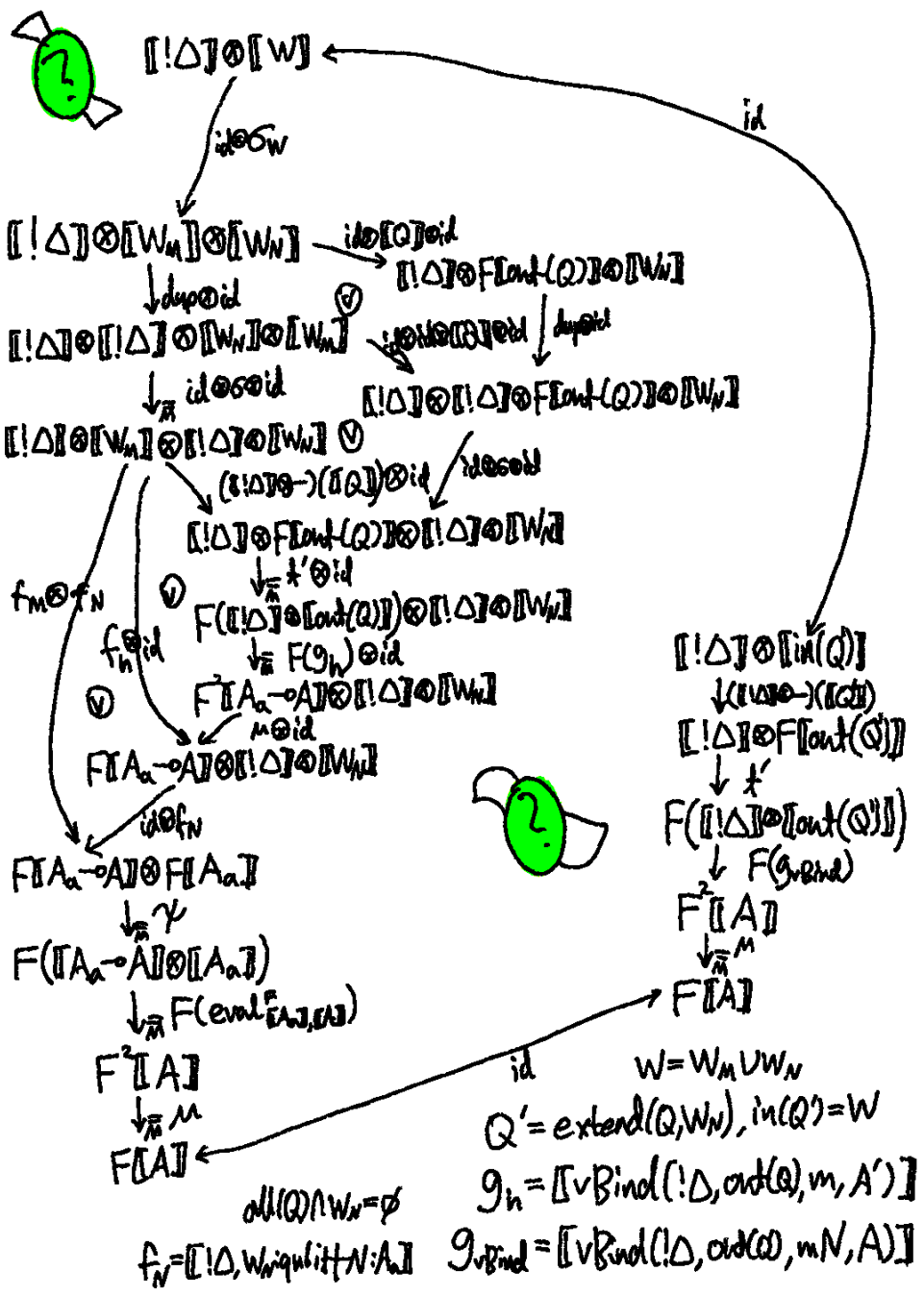
$$f_h = \llbracket v \text{Bind}(!\Delta, w_m, M, A') \rrbracket$$

$$g_h = \llbracket v \text{Bind}(!\Delta, out(Q), m, A') \rrbracket$$

Then, we show the following commutes:



$$W = W_m \cup W_n, Q' = \text{extend}(Q, W_n)$$



Proof of the lemma. 

Induction on Q:

$$\textcircled{1} Q = \varepsilon(X), \quad Q' = \text{extend}(Q, W_N) = \varepsilon(XUW_N)$$

since $(\varepsilon(W), M) \rightarrow (Q, n)$, $X = W_M$, $\text{in}(Q') = W = W_M U W_N$.

$$\text{Then from } \frac{X \cap \text{FV}(\Delta) = \emptyset \quad !\Delta, (W_M : \text{qubit}) \vdash m : A'}{\text{vBind}(!\Delta, X, m, A')}$$

$$\text{and } \frac{X \cap \text{FV}(\Delta) = \emptyset \quad \frac{!\Delta, W_N : \text{qubit} \vdash m : A' \quad !\Delta, W_N : \text{qubit} \vdash N : A_n}{!\Delta, (W : \text{qubit}) \vdash m : A}}{\text{vBind}(!\Delta, W, m, N, A)}$$

Let $g_m = [!\Delta, W_M : \text{qubit} \vdash m : A']$.

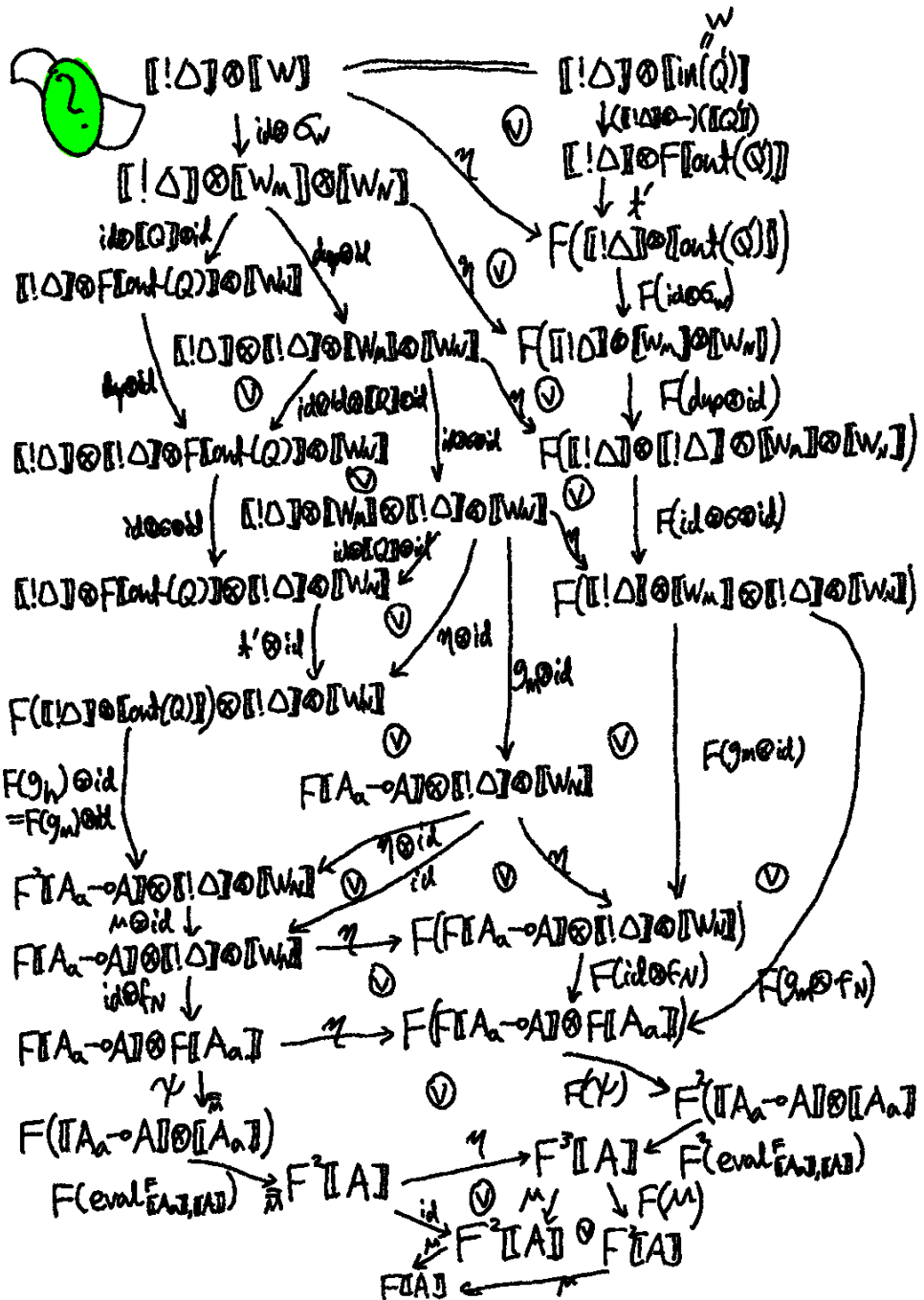
$$g_h = [\text{vBind}(!\Delta, X, m, A')] = [!\Delta] \otimes [X] \xrightarrow{g_m} F[A']$$

$$g_{\text{vBind}} = [\text{vBind}(!\Delta, W, m, N, A)]$$

$$= [!\Delta] \otimes [W] \xrightarrow{\text{id} \otimes \text{ev}_N} [!\Delta] \otimes [W_N] \otimes [W_N] \xrightarrow{\text{dup} \otimes \text{id}} [!\Delta] \otimes [!\Delta] \otimes [W_N] \otimes [W_N]$$

$$F[A_n \multimap A] \otimes F[A_n] \xleftarrow{g_m \otimes f_N} [!\Delta] \otimes [W_N] \otimes [!\Delta] \otimes [W_N] \xleftarrow{\text{id} \otimes \text{ev}_N}$$

$$F([A_n \multimap A] \otimes [A_n]) \xrightarrow{\text{eval}_{F[A_n], [A]}} F^2[A] \xrightarrow{\mu} F[A]$$



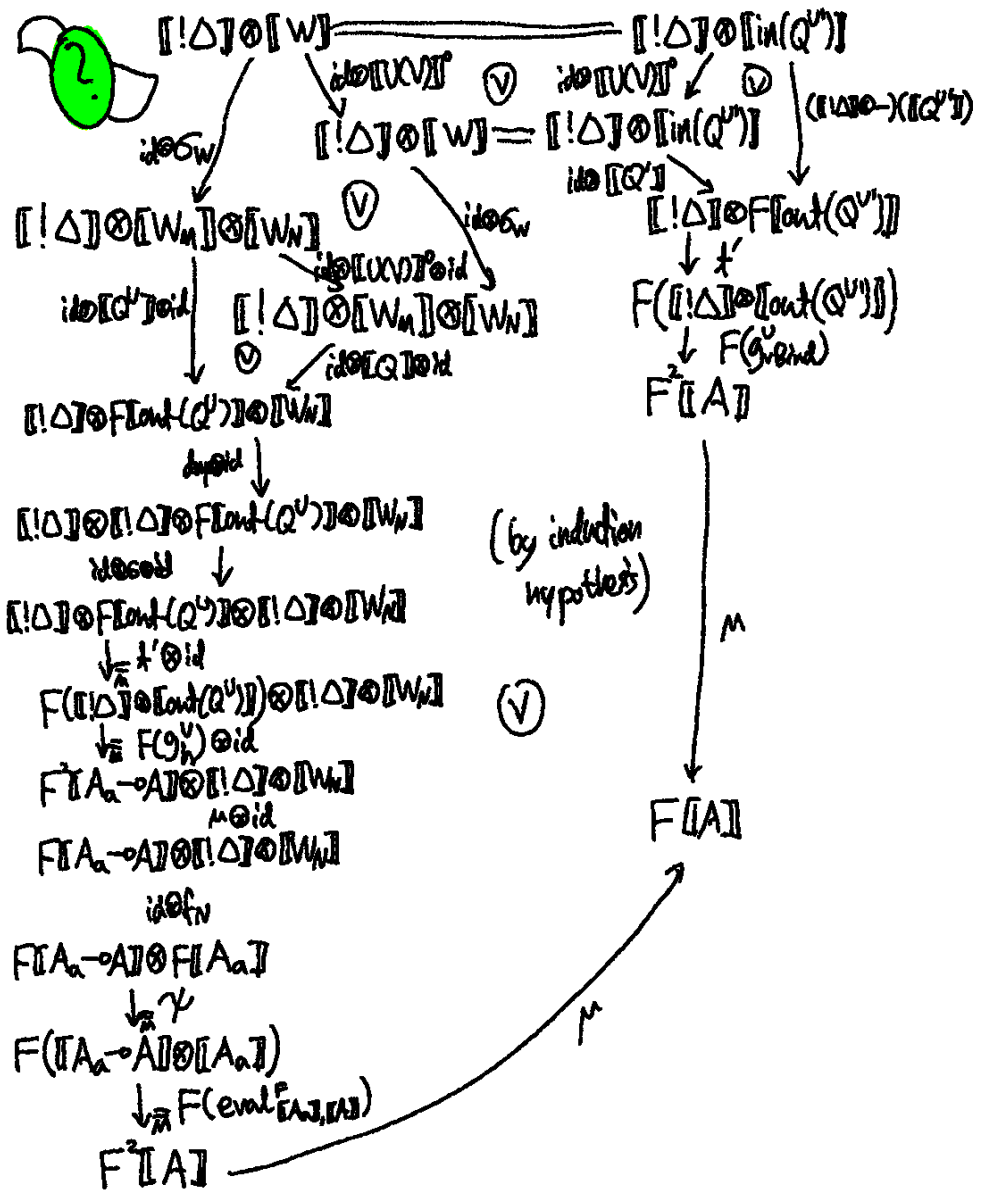
$$\textcircled{2} \quad Q^v = U(V) Q$$

$$\text{out}(Q) = \text{out}(Q^v), \quad \llbracket Q^v \rrbracket = \llbracket Q \rrbracket \circ \llbracket U(V) \rrbracket^o$$

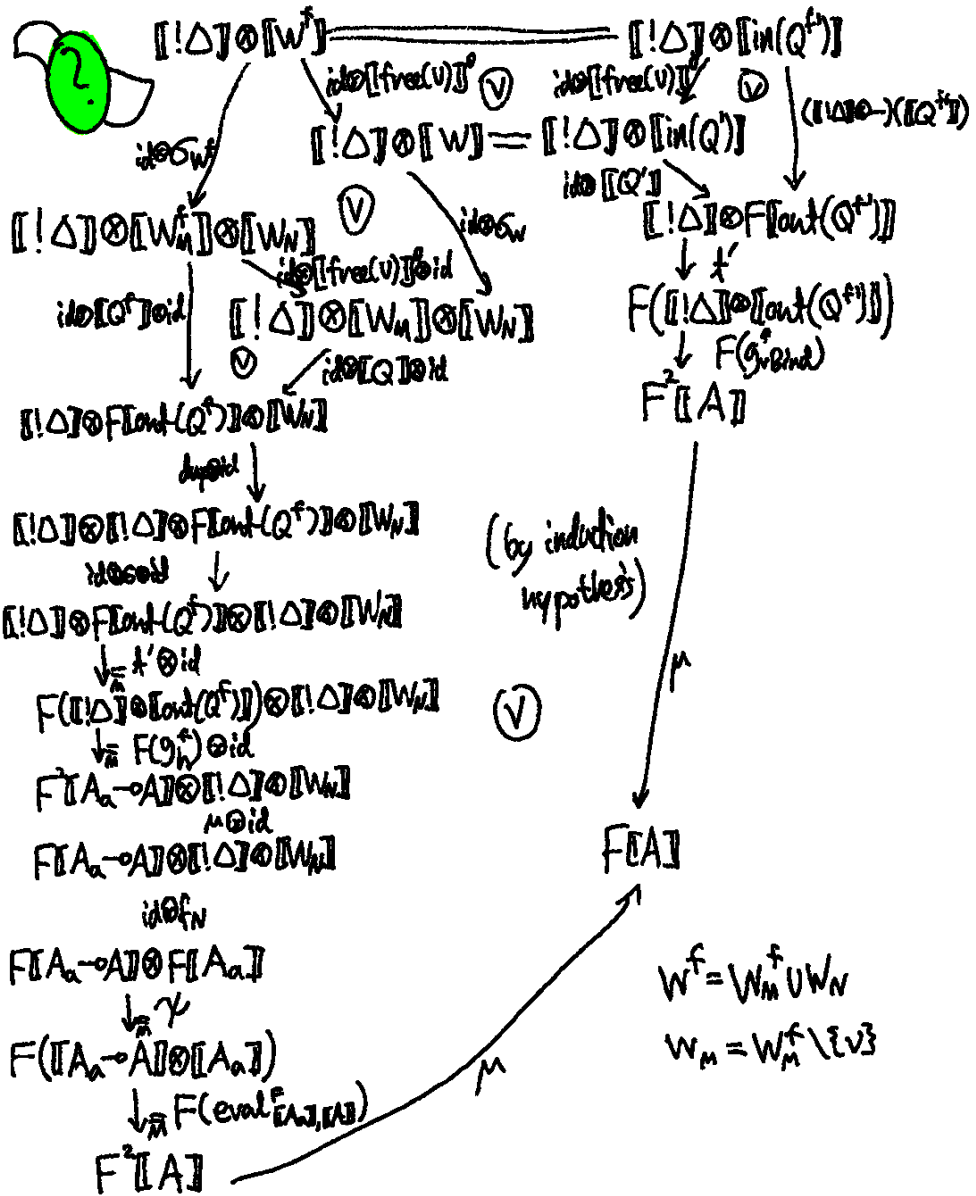
$$\begin{aligned} Q^{v'} &= \text{extend}(Q^v, w_N) & \llbracket Q^{v'} \rrbracket &= \llbracket Q^v \rrbracket \circ \llbracket U(V) \rrbracket^o \\ &= U(V) \text{extend}(Q, w_N) & \text{in}(Q^{v'}) &= \text{in}(Q^v) \cup w_N = w \\ &= U(V) Q' & \text{out}(Q^{v'}) &= \text{out}(Q') \end{aligned}$$

$$\begin{aligned} \text{and } g_n^v &= \llbracket v\text{Bind}(!\Delta, \text{out}(Q), m, A') \rrbracket \\ &= \llbracket v\text{Bind}(!\Delta, \text{out}(Q^v), m, A') \rrbracket = g_n \end{aligned}$$

$$\begin{aligned} g_{v\text{bind}}^v &= \llbracket v\text{Bind}(!\Delta, \text{out}(Q), mN, A) \rrbracket \\ &= \llbracket v\text{Bind}(!\Delta, \text{out}(Q^v), mN, A) \rrbracket = g_{v\text{bind}} \end{aligned}$$



$$\begin{aligned}
\textcircled{3} \quad Q^f &= \text{free } v \ Q & \text{in}(Q^f) &= \text{in}(Q) \cup \{v\} \\
\text{out}(Q) &= \text{out}(Q^f), & \llbracket Q^f \rrbracket &= \llbracket Q \rrbracket \circ \llbracket \text{free}(v) \rrbracket^\circ \\
Q^{f'} &= \text{extend}(Q^f, w_N) & \llbracket Q^{f'} \rrbracket &= \llbracket Q^f \rrbracket \circ \llbracket \text{free}(v) \rrbracket^\circ \\
&= \text{free } v \ \text{extend}(Q, w_N) & \text{in}(Q^{f'}) &= \text{in}(Q^f) \cup w_N = w^f \\
&= \text{free } v \ Q' & \text{out}(Q^{f'}) &= \text{out}(Q') \\
\text{and } g_n^f &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q), m, A') \rrbracket \\
&= \llbracket v \text{Bind}(!\Delta, \text{out}(Q^f), m, A') \rrbracket = g_n \\
g_{v \text{bind}}^f &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q'), m \setminus N, A) \rrbracket \\
&= \llbracket v \text{Bind}(!\Delta, \text{out}(Q^{f'}), m \setminus N, A) \rrbracket = g_{v \text{bind}}
\end{aligned}$$



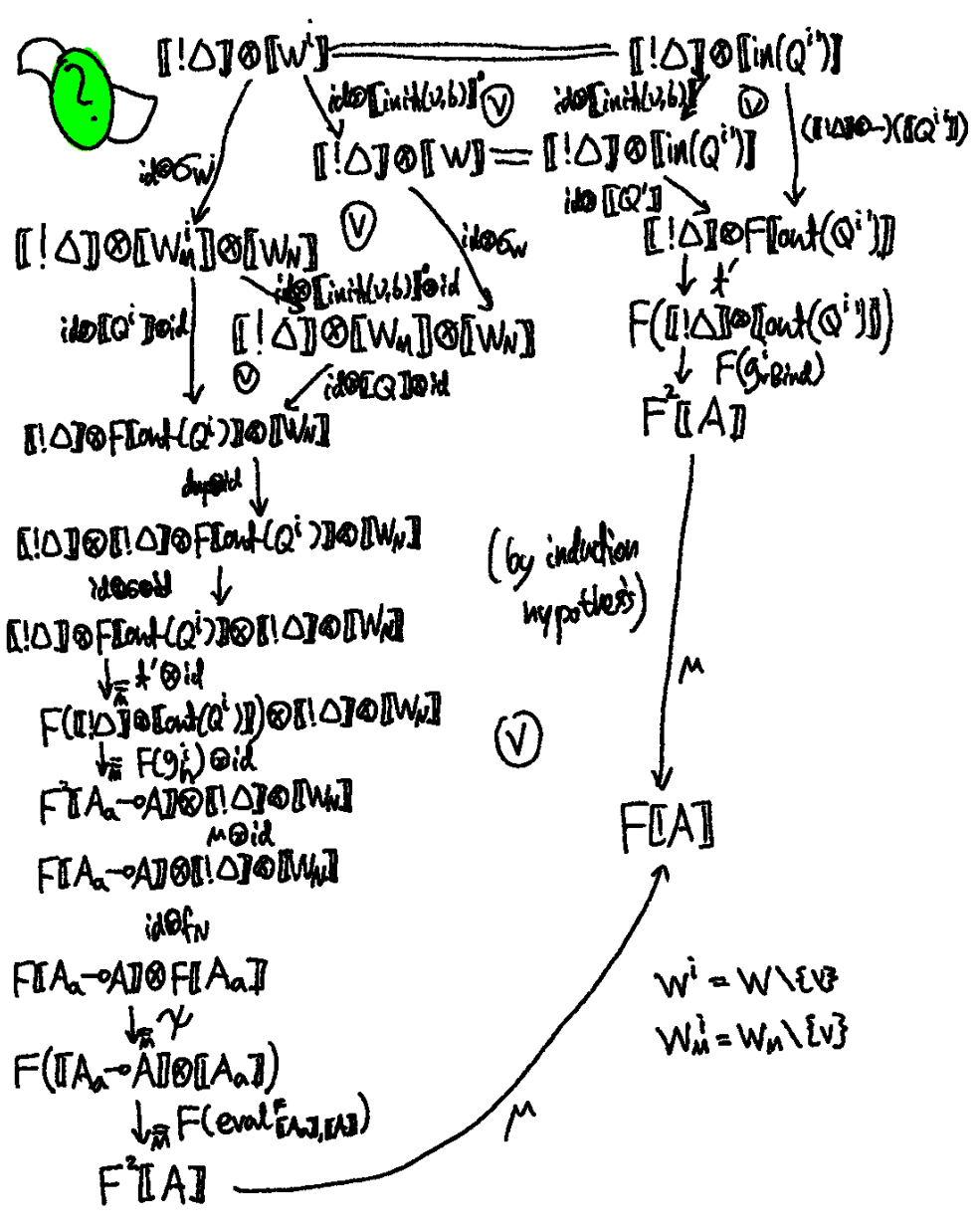
$$\textcircled{4} \quad Q^i = \text{init } b \vee Q \quad \text{in}(Q^i) = \text{in}(Q) \setminus \{v\}$$

$$\text{out}(Q) = \text{out}(Q^i), \quad \llbracket Q^i \rrbracket = \llbracket Q \rrbracket \circ \llbracket \text{init}(v, b) \rrbracket^\circ$$

$$\begin{aligned} Q^{i'} &= \text{extend}(Q^i, w_N) & \llbracket Q^{i'} \rrbracket &= \llbracket Q^i \rrbracket \circ \llbracket \text{init}(v, b) \rrbracket^\circ \\ &= \text{init } b \vee \text{extend}(Q, w_N) & \text{in}(Q^{i'}) &= \text{in}(Q^i) \cup w_N = w^i \\ &= \text{init } b \vee Q' & \text{out}(Q^{i'}) &= \text{out}(Q') \end{aligned}$$

$$\text{and } g_{in}^i = \llbracket v \text{Bind}(!\Delta, \text{out}(Q^i), m, A') \rrbracket \\ = \llbracket v \text{Bind}(!\Delta, \text{out}(Q), m, A') \rrbracket = g_{in}$$

$$\begin{aligned} g_{v \text{bind}}^i &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q^{i'}), m, A) \rrbracket \\ &= \llbracket v \text{Bind}(!\Delta, \text{out}(Q'), m, A) \rrbracket = g_{v \text{bind}} \end{aligned}$$



$$\textcircled{5} Q^m = \text{meas } \nu Q_1, Q_2, m = [m_1, m_2]$$

$$\text{out}(Q^m) = [\text{out}(Q_1), \text{out}(Q_2)], \text{in}(Q^m) = \text{in}(Q_{12})$$

$$\begin{aligned} \llbracket Q^m \rrbracket &= \llbracket \text{in}(\text{meas } \nu Q_1, Q_2) \rrbracket \xrightarrow{\text{bit}} F(\llbracket \text{in}(\text{meas } \nu Q_1, Q_2) \rrbracket \\ &\quad + \llbracket \text{in}(\text{meas } \nu Q_1, Q_2) \rrbracket) \\ &\quad \downarrow F(\llbracket Q_1 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket \\ &\quad \quad + \llbracket Q_2 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket) \end{aligned}$$

$$Q^{m'} = \text{extend}(Q^m, W_N)$$

$$= \text{meas } \nu Q'_1, Q'_2$$

$$Q'_1 = \text{extend}(Q_1, W_N)$$

$$Q'_2 = \text{extend}(Q_2, W_N)$$

$$\text{in}(Q^{m'}) = \text{in}(Q^m) \cup W_N = W$$

$$\text{out}(Q^{m'}) = [\text{out}(Q'_1), \text{out}(Q'_2)]$$

$$F(F\llbracket \text{out}(Q_1) \rrbracket + F\llbracket \text{out}(Q_2) \rrbracket)$$

$$\downarrow F[\text{merge}]$$

$$F^2(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket)$$

$$\downarrow M$$

$$F(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket)$$

$$\llbracket Q^{m'} \rrbracket = \llbracket \text{in}(Q^{m'}) \rrbracket \xrightarrow{\text{bit}} F(\llbracket \text{in}(Q^{m'}) \rrbracket + \llbracket \text{in}(Q^{m'}) \rrbracket)$$

$$\downarrow F(\llbracket Q'_1 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket \\ + \llbracket Q'_2 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket)$$

$$F(F\llbracket \text{out}(Q'_1) \rrbracket + F\llbracket \text{out}(Q'_2) \rrbracket)$$

$$\downarrow F[\text{merge}]$$

$$F^2(\llbracket \text{out}(Q'_1) \rrbracket + \llbracket \text{out}(Q'_2) \rrbracket)$$

$$\downarrow M$$

$$F(\llbracket \text{out}(Q'_1) \rrbracket + \llbracket \text{out}(Q'_2) \rrbracket)$$

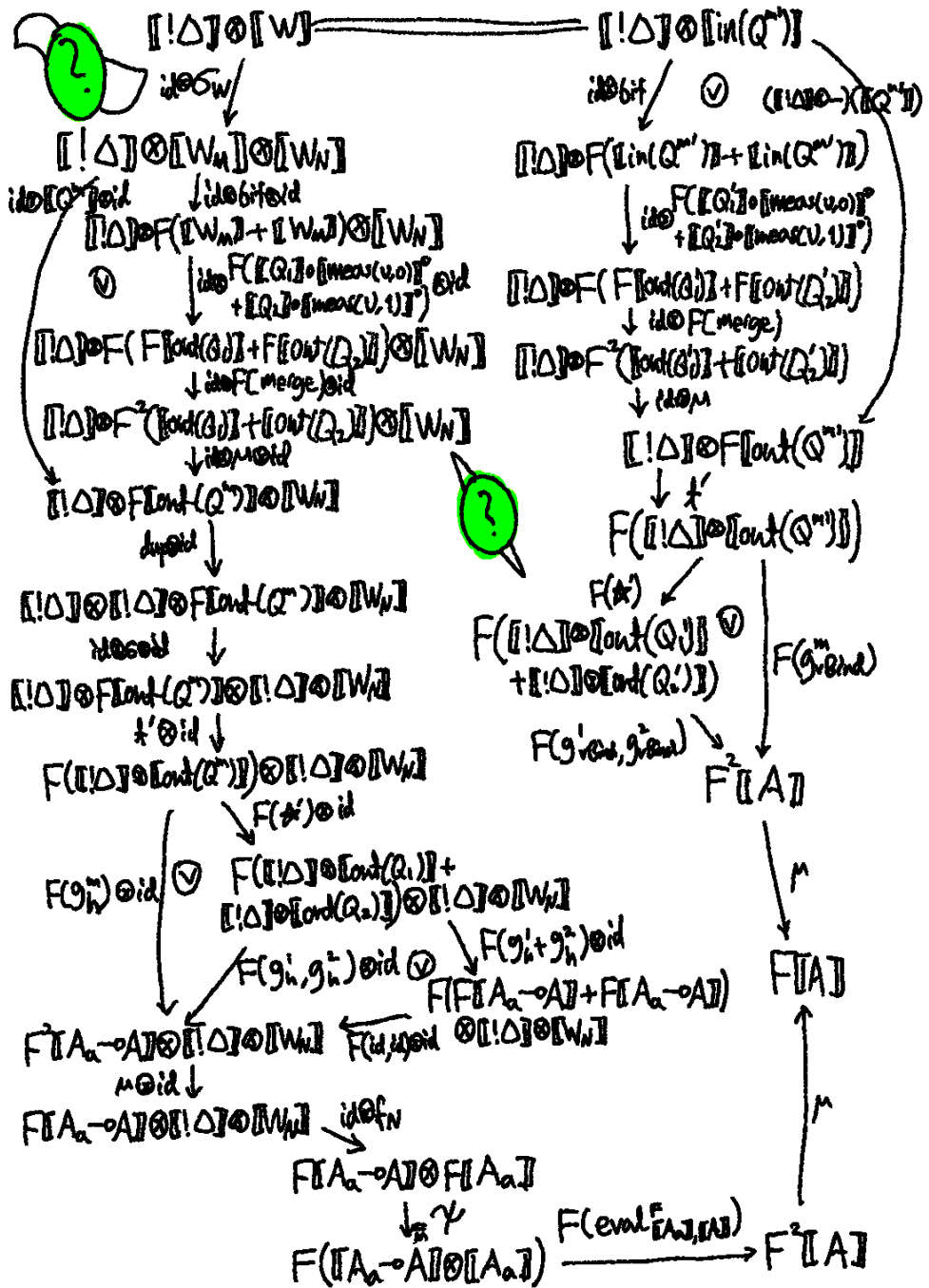
$$\text{and } g_h^m = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q^m), m, A') \rrbracket$$

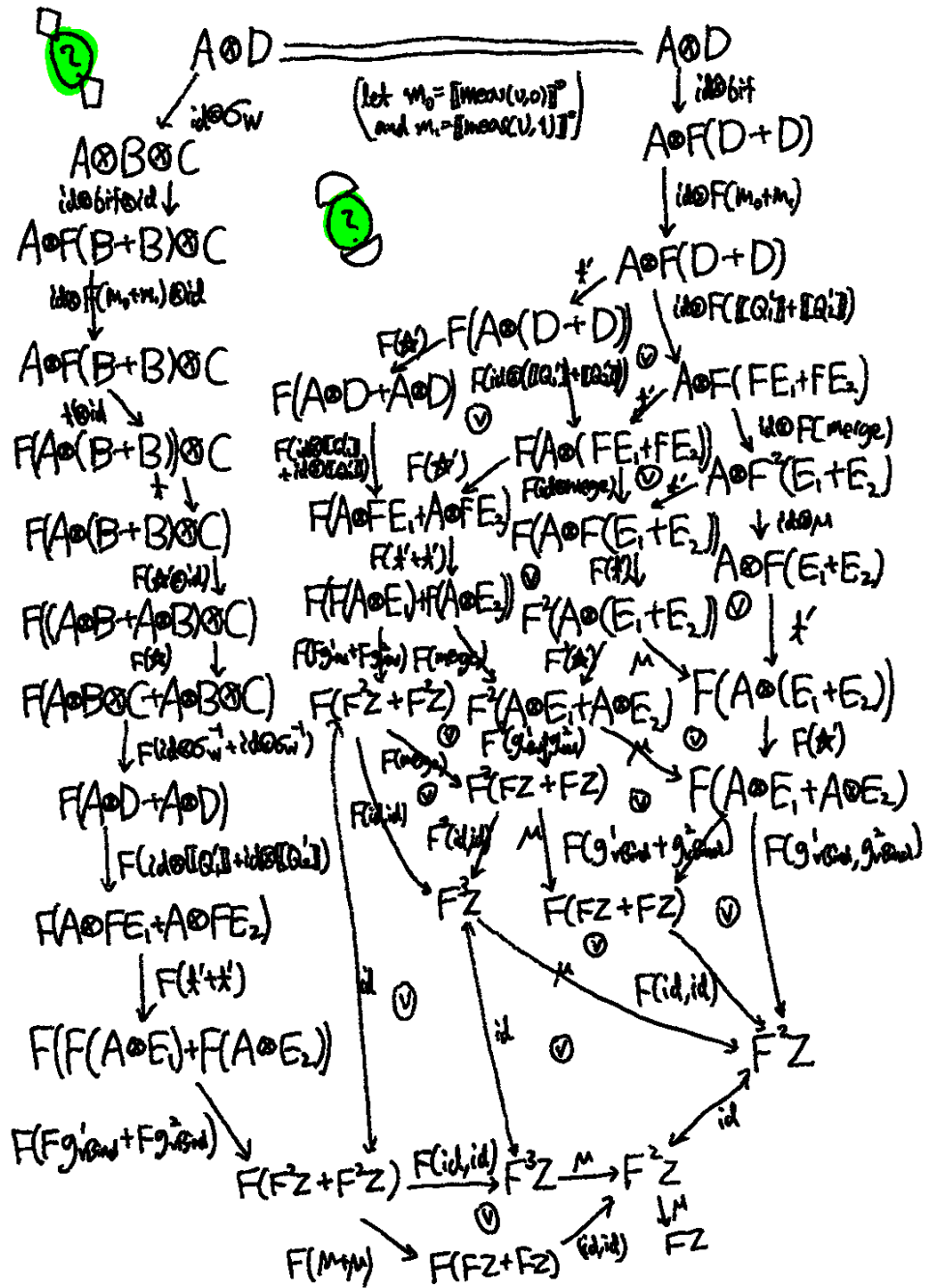
$$= \llbracket !\Delta \rrbracket \otimes (\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \xrightarrow{*} \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q_1) \rrbracket + \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q_2) \rrbracket \xrightarrow{(g_1^{\text{end}}, g_2^{\text{end}})} \llbracket A' \rrbracket$$

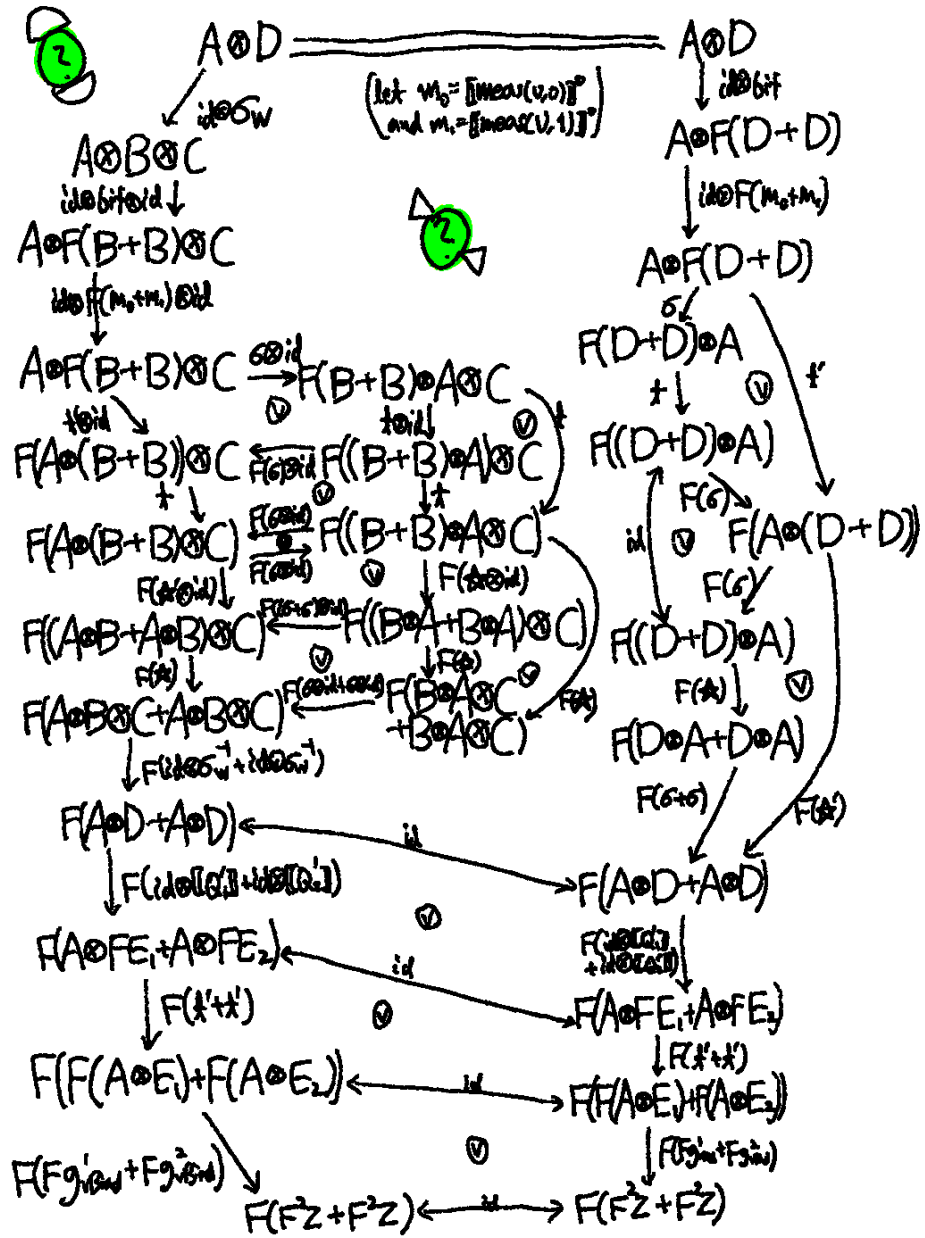
$$g_{\nu \text{bind}}^m = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q^{m'}), m, N, A) \rrbracket$$

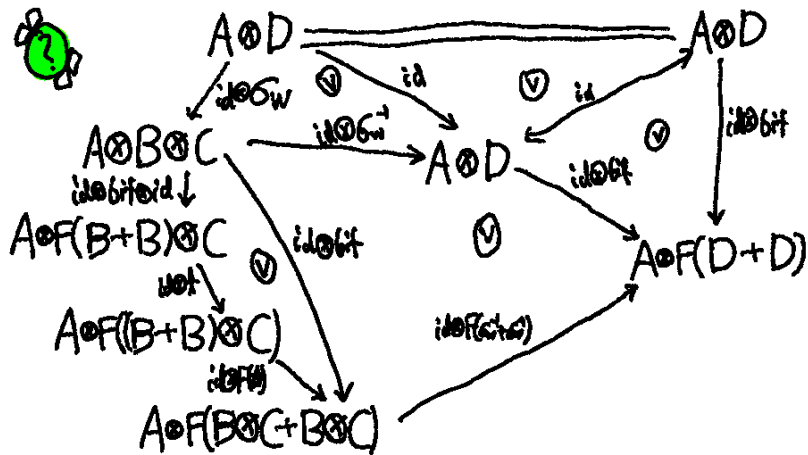
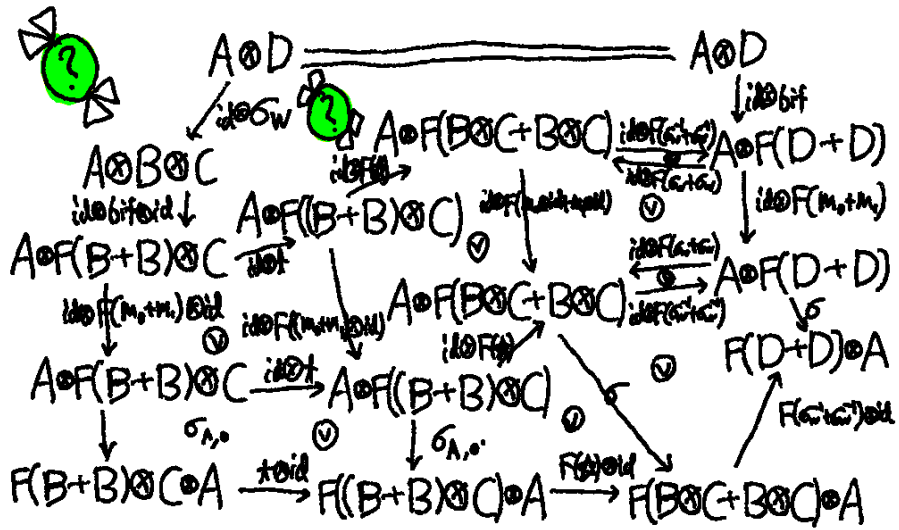
$$= \llbracket !\Delta \rrbracket \otimes (\llbracket \text{out}(Q'_1) \rrbracket + \llbracket \text{out}(Q'_2) \rrbracket) \xrightarrow{*} \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q'_1) \rrbracket + \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q'_2) \rrbracket$$

$$\downarrow (g_1^{\text{end}}, g_2^{\text{end}}) \\ \llbracket A \rrbracket$$









A.8 . Congruence: on the left of application

$$\frac{(\mathcal{E}(W_M), M) \rightarrow (Q, m) \quad \text{all}(Q) \cap W_N = \emptyset}{(\mathcal{E}(W_M \cup W_N), NM) \rightarrow (\text{extend}(Q, W_N), Nm)}$$

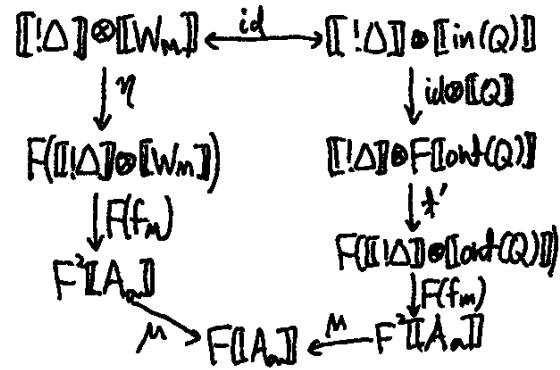
$$\begin{aligned} [[\Delta \vdash (\mathcal{E}(W_M \cup W_N), NM) : A]] &= [[[\mathcal{E}(W_M \cup W_N)], f_{NM}]] \\ &= [[! \Delta] \circ [W_M \cup W_N]] \xrightarrow{\text{id}[\mathcal{E}(W_M \cup W_N)]} [[! \Delta] \circ F[W_M \cup W_N]] \\ &\quad \searrow \eta \quad \circ \quad \downarrow \sharp' \\ &\quad \quad \quad F[[! \Delta] \circ [W_M \cup W_N]] \\ &\quad \quad \quad \downarrow F(f_{NM}) \\ &\quad \quad \quad F[A] \\ &\quad \quad \quad \downarrow \mu \\ &\quad \quad \quad F[A] \end{aligned}$$

$$f_{NM} = [[\text{vBind}(! \Delta, W_M \cup W_N, NM, A)]]$$

$$\begin{aligned} [[\Delta \vdash (\text{extend}(Q, W_N), Nm) : A]] &\quad \text{in}(\text{extend}(Q, W_N)) \\ &= [[[\text{extend}(Q, W_N)], f_{Nm}]] \quad = \text{in}(Q) \cup W_N \\ &= [[! \Delta] \circ [\text{in}(Q) \cup W_N]] \xrightarrow{\text{id}[\text{extend}(Q, W_N)]} [[! \Delta] \circ F[\text{out}(\text{extend}(Q, W_N))]] \\ &\quad \quad \quad \downarrow \sharp' \\ &\quad \quad \quad F[[! \Delta] \circ [\text{out}(\text{extend}(Q, W_N))]] \\ &\quad \quad \quad \downarrow F(f_{Nm}) \\ &\quad \quad \quad F[A] \\ &\quad \quad \quad \downarrow \mu \\ &\quad \quad \quad F[A] \end{aligned}$$

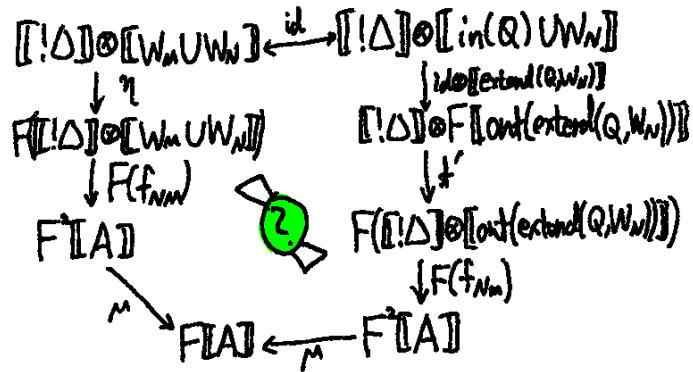
$$f_{Nm} = [[\text{vBind}(! \Delta, \text{out}(\text{extend}(Q, W_N)), Nm, A)]]$$

From the induction hypothesis:



where $f_n = [! vBind(! \Delta, W_n, M, A_n)]$ and $f_m = [! vBind(! \Delta, out(Q), m, A_n)]$.

Then we show the following:

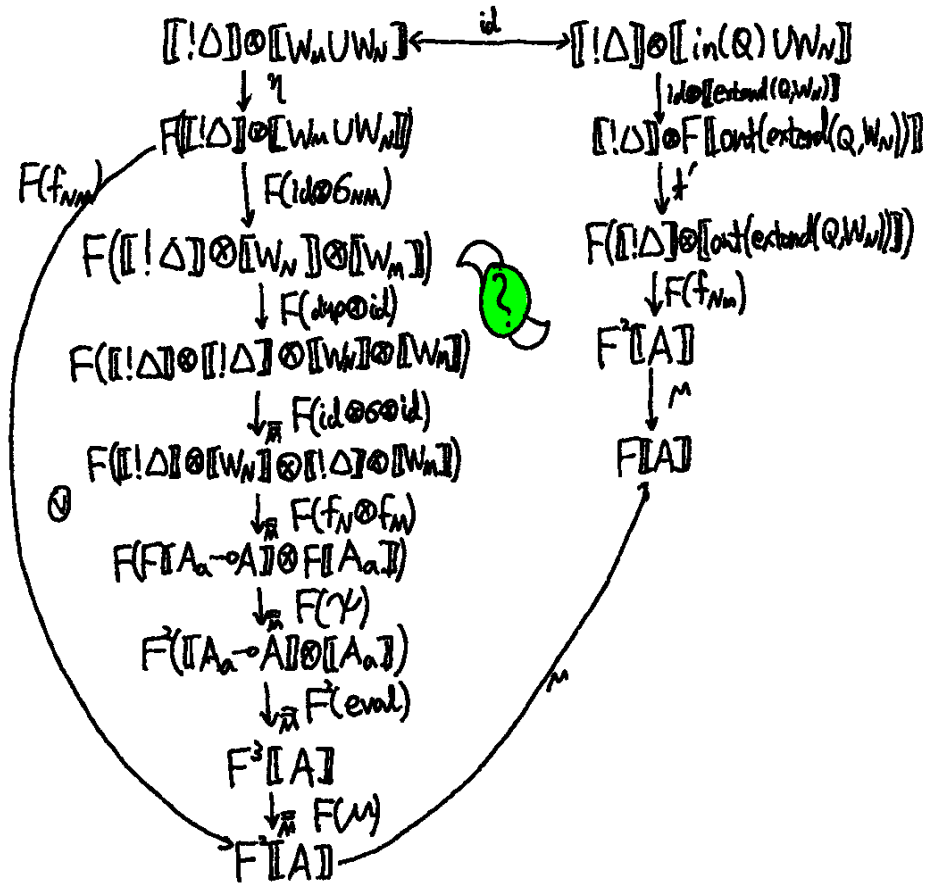


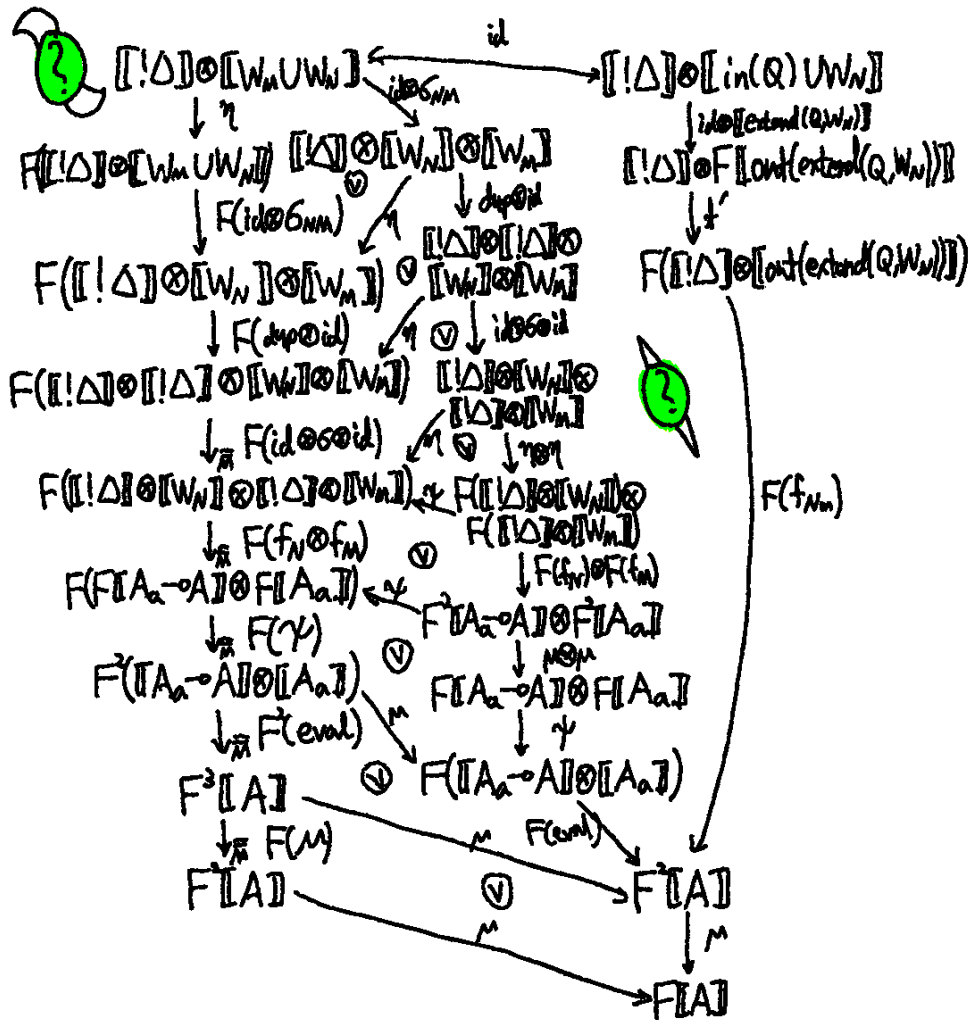
Since

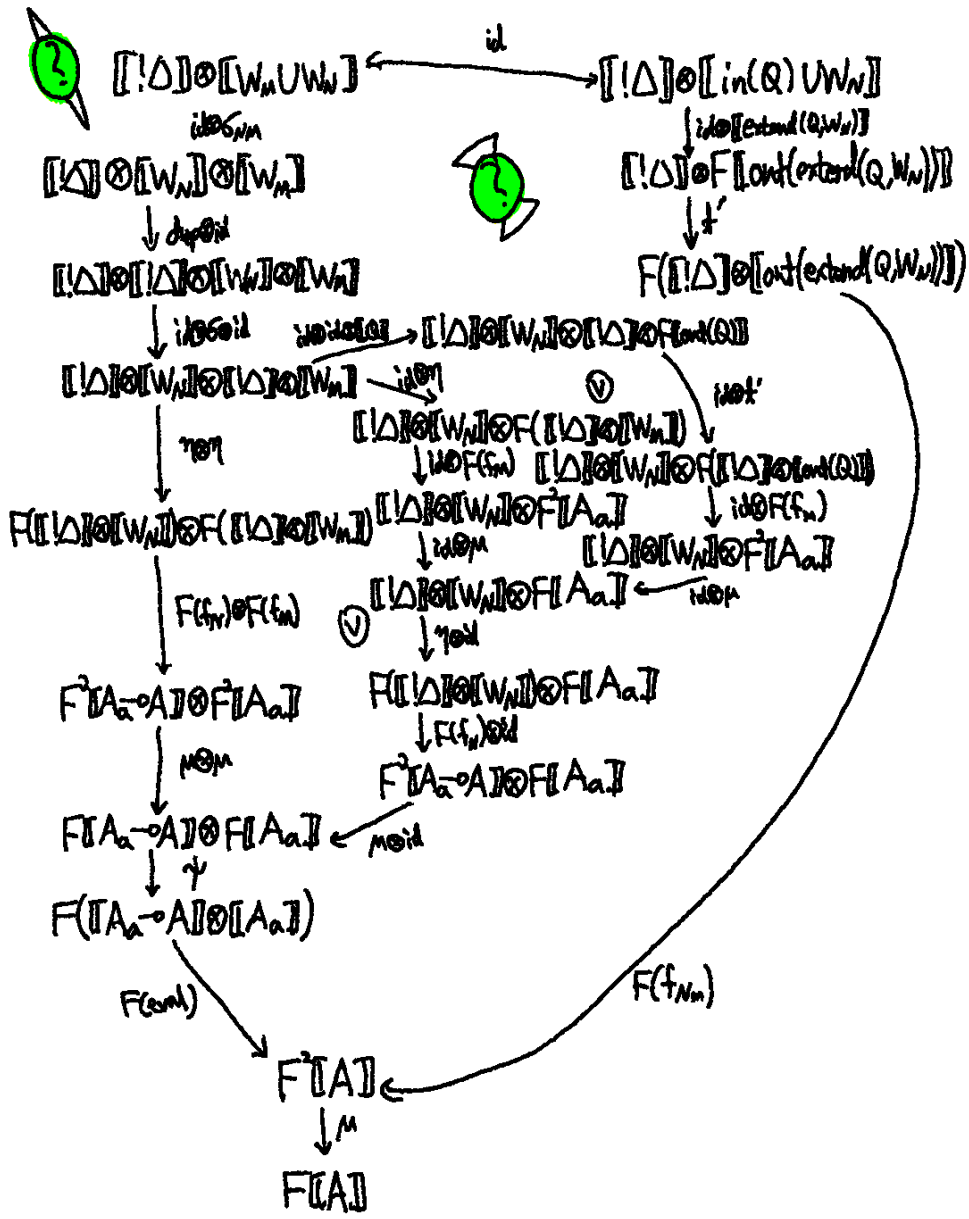
$$\begin{aligned}
 f_{NM} &= [! \Delta] \otimes [W_M U W_N] \\
 &\quad \downarrow \text{id} \otimes \sigma_{NM} \\
 &= [! \Delta] \otimes [W_N] \otimes [W_M] \\
 &\quad \downarrow \text{id} \otimes \text{id} \\
 &= [! \Delta] \otimes [! \Delta] \otimes [W_N] \otimes [W_M] \\
 &\quad \downarrow \text{id} \otimes \text{id} \\
 &= [! \Delta] \otimes [W_N] \otimes [! \Delta] \otimes [W_M] \\
 &\quad \downarrow f_N \otimes f_M \quad f_N = [! \Delta, W_N \text{ qubit} \vdash N : A_n \multimap A] \\
 &= F[A_n \multimap A] \otimes F[A_n] \\
 &\quad \downarrow \gamma \\
 &= F([A_n \multimap A] \otimes [A_n]) \\
 &\quad \downarrow F(\text{eval}_{[A_n], [A]}) \\
 &= F^2[A] \\
 &\quad \downarrow \mu \\
 &= F[A]
 \end{aligned}$$



It reduces to show the following:







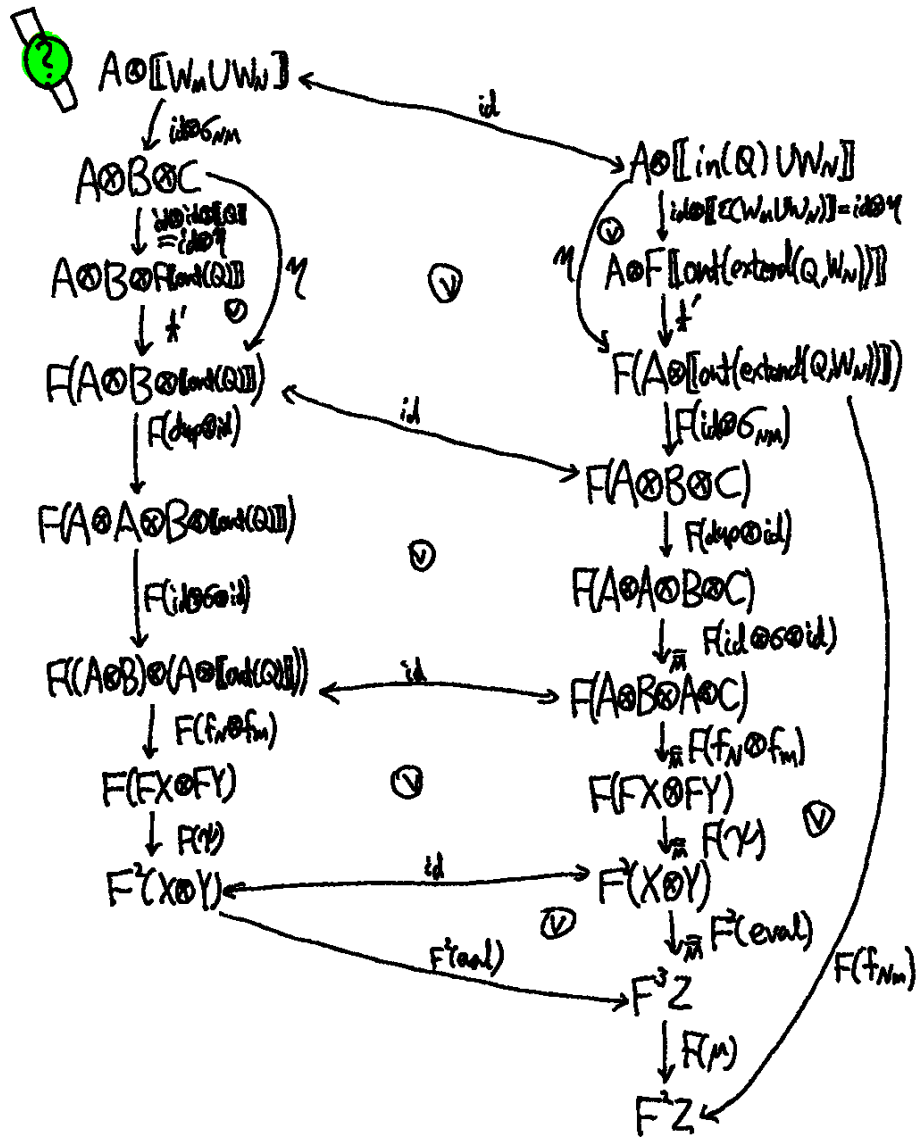




Proof by induction on Q .

$$\textcircled{1} Q = E(W_N), \text{ extend}(Q, W_M) = E(W_M U W_N)$$

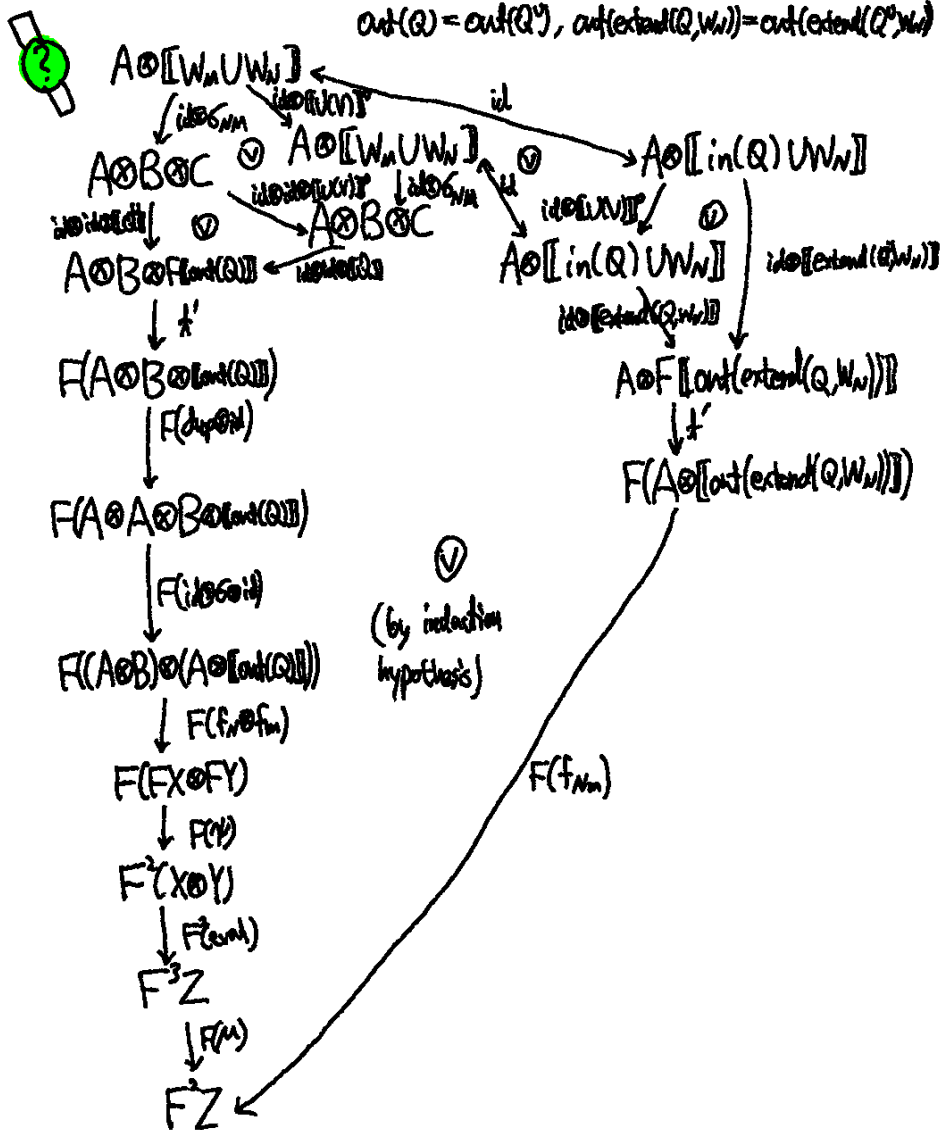
$$\begin{aligned} f_{N_M} &= [\nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), N_M, A)] \\ &= [!\Delta] \otimes [W_M U W_N] \\ &\quad \downarrow \text{id} \otimes \sigma_{M,N} \\ &= [!\Delta] \otimes [W_N] \otimes [W_M] \\ &\quad \downarrow \text{dup} \otimes \text{id} \\ &= [!\Delta] \otimes [!\Delta] \otimes [W_N] \otimes [W_M] \\ &\quad \downarrow \text{id} \otimes \sigma_{M,N} \\ &= [!\Delta] \otimes [W_N] \otimes [!\Delta] \otimes [W_M] \\ &\quad \downarrow f_N \otimes f_M \\ &= F[A_N \multimap A] \otimes F[A_M] \\ &\quad \downarrow \gamma \\ &= F([A_N \multimap A] \otimes [A_M]) \\ &\quad \downarrow F(\text{eval}_{[A_N], [A_M]}) \\ &= F^2[A] \\ &\quad \downarrow \mu \\ &= F[A] \end{aligned}$$



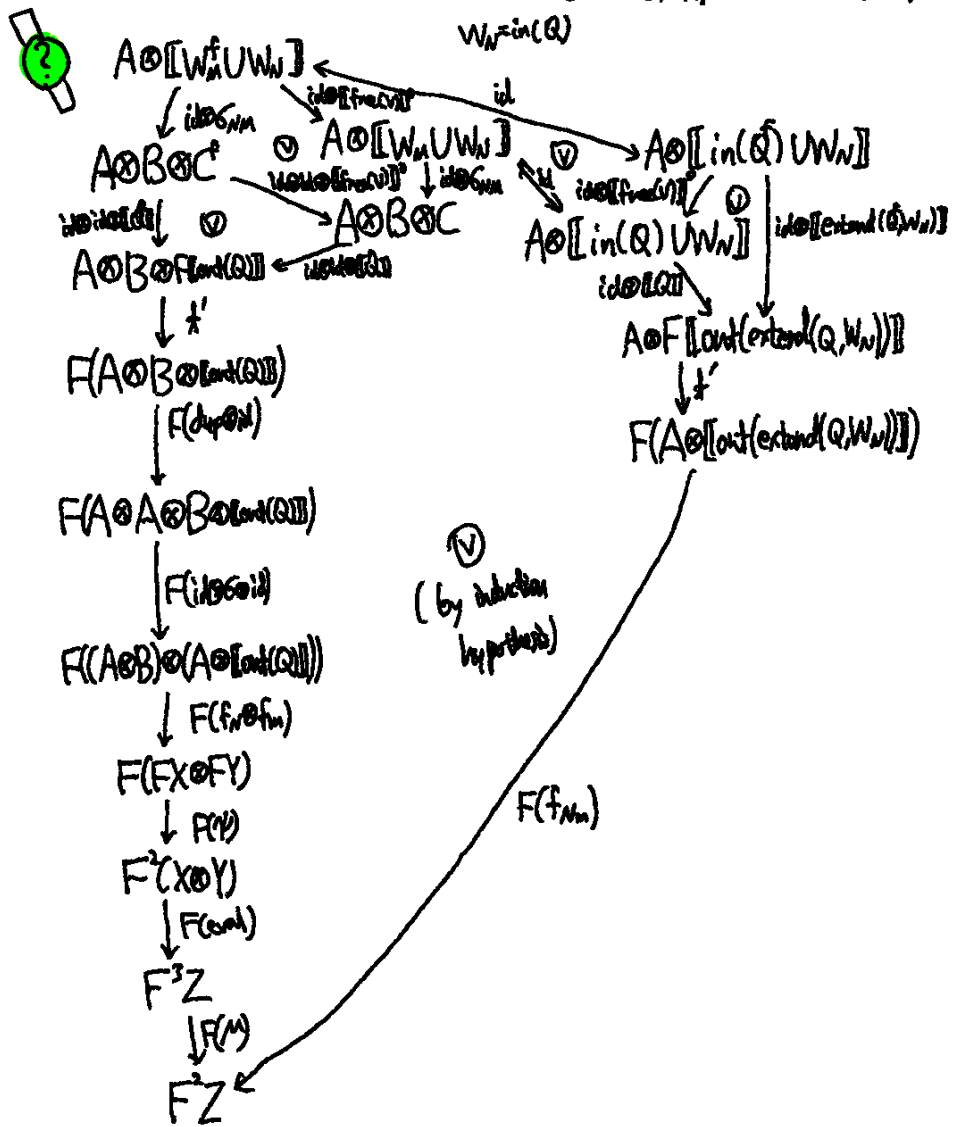
$$\textcircled{2} Q^u = U(V)Q, \quad [Q^u] = [Q] \cdot [U(V)]^*$$

$$[\text{extend}(Q^u, w)] = [\text{extend}(Q, w)] \cdot [U(V)]^*$$

$$\text{out}(Q) = \text{out}(Q^u), \quad \text{out}(\text{extend}(Q, w)) = \text{out}(\text{extend}(Q^u, w))$$

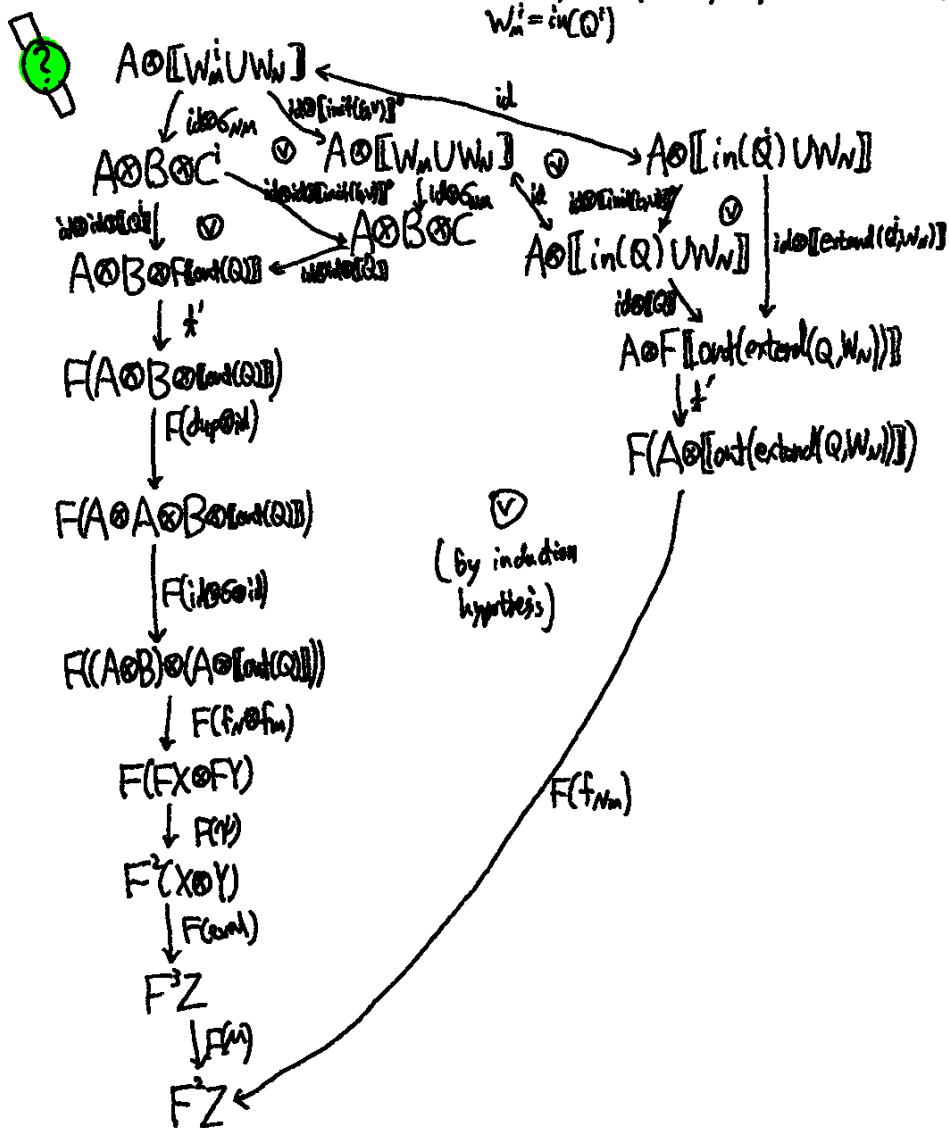


③ $Q^f = \text{free } v Q, [Q^f] = [Q] \cdot [\text{free}(V)]^f$
 $[\text{extend}(Q^f, W_N)] = [\text{extend}(Q, W_N)] \cdot [\text{free}(W)]^f$
 $\text{out}(Q) = \text{out}(Q^f), \text{out}(\text{extend}(Q, W_N)) = \text{out}(\text{extend}(Q^f, W_N))$



$$\textcircled{4} Q^i = \text{init } b \vee Q$$

$$\begin{aligned} [Q^i] &= [Q] \cdot [\text{init}(b, v)]^* \\ [\text{extend}(Q^i, w_N)] &= [\text{extend}(Q, w_N)] \cdot [\text{init}(b, v)]^* \\ \text{out}(Q^i) &= \text{out}(Q), \text{out}(\text{extend}(Q^i, w_N)) = \text{out}(\text{extend}(Q, w_N)) \\ w_N^i &= \text{in}(Q^i) \end{aligned}$$



$$\textcircled{5} Q^m = \text{mers } v Q_1 Q_2$$

$$\text{Let } (Q^m)' = \text{extend}(Q^m, W_N)$$

$$Q_1' = \text{extend}(Q_1, W_N)$$

$$Q_2' = \text{extend}(Q_2, W_N)$$

$$\text{in}(Q^m) = \text{in}(Q_1) = \text{in}(Q_2)$$

$$\text{in}((Q^m)') = \text{in}(Q_1') = \text{in}(Q_2')$$

$$\text{out}(Q^m) = [\text{out}(Q_1), \text{out}(Q_2)]$$

$$\text{out}((Q^m)') = [\text{out}(Q_1'), \text{out}(Q_2')]$$

$$[[Q^m]] = [[\text{in}(\text{mers } v Q_1 Q_2)]]$$

$$\downarrow \text{bit}$$

$$F([\text{in}(\text{mers } v Q_1 Q_2)] + [\text{in}(\text{mers } v Q_1 Q_2)])$$

$$\downarrow F([Q_1] \circ [\text{mers}(v, 0)]^* + [Q_2] \circ [\text{mers}(v, 1)]^*)$$

$$F(F[\text{out}(Q_1)] + F[\text{out}(Q_2)])$$

$$\downarrow \text{FC merge}$$

$$F^2([\text{out}(Q_1)] + [\text{out}(Q_2)])$$

$$\downarrow M$$

$$F([\text{out}(Q_1)] + [\text{out}(Q_2)])$$

$$f_m = [[!\Delta]] \circ ([\text{out}(Q_1)] + [\text{out}(Q_2)])$$

$$\downarrow \star'$$

$$[[!\Delta]] \circ ([\text{out}(Q_1)] + [[!\Delta]] \circ [\text{out}(Q_2)])$$

$$\downarrow (f_m^1, f_m^2)$$

$$F[A]$$

$$f_{m'} = [[!\Delta]] \circ ([\text{out}(Q_1')] + [\text{out}(Q_2')])$$

$$\downarrow \star'$$

$$[[!\Delta]] \circ [\text{out}(Q_1')] + [[!\Delta]] \circ [\text{out}(Q_2')]$$

$$\downarrow (f_{m'}^1, f_{m'}^2)$$

$$F[A]$$

$$[[Q^m]'] = [[\text{in}((Q^m)')]]$$

$$\downarrow \text{bit}$$

$$F([\text{in}((Q^m)')] + [\text{in}((Q^m)')])$$

$$\downarrow F([Q_1'] \circ [\text{mers}(v, 0)]^* + [Q_2'] \circ [\text{mers}(v, 1)]^*)$$

$$F(F[\text{out}(Q_1')] + F[\text{out}(Q_2')])$$

$$\downarrow \text{FC merge}$$

$$F^2([\text{out}(Q_1')] + [\text{out}(Q_2')])$$

$$\downarrow M$$

$$F([\text{out}(Q_1')] + [\text{out}(Q_2')])$$

A.9 . Congruence: on the left of a pair

$$\frac{(\varepsilon(W_M), M) \rightarrow (Q, m) \quad \text{all}(Q) \wedge W_N = \emptyset}{(\varepsilon(W_M \cup W_N), \langle M, N \rangle) \rightarrow (\text{extend}(Q, W_N), \langle m, N \rangle)}$$

$$\begin{aligned} \llbracket !\Delta \vdash (\varepsilon(W_M \cup W_N), \langle M, N \rangle) : A \rrbracket &= \llbracket ([\varepsilon(W_M \cup W_N)], f_{\langle M, N \rangle}) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket [W_M \cup W_N] \rrbracket \xrightarrow{\eta} F(\llbracket !\Delta \rrbracket \otimes \llbracket [W_M \cup W_N] \rrbracket) \xrightarrow{F(f_{\langle M, N \rangle})} F^2[A] \xrightarrow{\mu} F[A] \end{aligned}$$

$$f_{\langle M, N \rangle} = \llbracket \nu \text{Bind}(!\Delta, W_M \cup W_N, \langle M, N \rangle, A) \rrbracket$$

$$\begin{aligned} \llbracket !\Delta \vdash (\text{extend}(Q, W_N), \langle m, N \rangle) : A \rrbracket &= \llbracket ([\text{extend}(Q, W_N)], f_{\langle m, N \rangle}) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket [\text{in}(Q) \cup W_N] \rrbracket \xrightarrow{\text{id} \otimes [\text{extend}(Q, W_N)]} \llbracket !\Delta \rrbracket \otimes F(\llbracket [\text{out}(Q), W_N] \rrbracket) \end{aligned}$$

$$\begin{aligned} &\downarrow \eta' \\ &F(\llbracket !\Delta \rrbracket \otimes \llbracket [\text{out}(Q), W_N] \rrbracket) \\ &\downarrow F(f_{\langle m, N \rangle}) \\ &F^2[A] \\ &\downarrow \mu \\ &F[A] \end{aligned}$$

$$f_{\langle m, N \rangle} = \llbracket \nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \langle m, N \rangle, A) \rrbracket$$

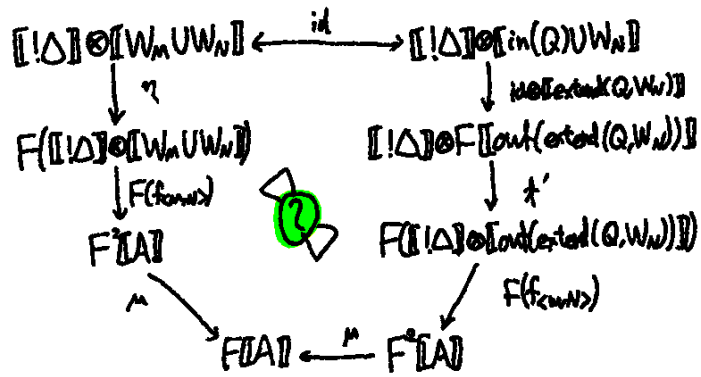
From the induction hypothesis:

$$\begin{array}{ccc} \llbracket !\Delta \rrbracket \otimes \llbracket [W_M] \rrbracket & \xleftarrow{\quad} & \llbracket !\Delta \rrbracket \otimes \llbracket [\text{in}(Q)] \rrbracket \\ \downarrow \eta & & \downarrow \text{id} \otimes [Q] \\ F(\llbracket !\Delta \rrbracket \otimes \llbracket [W_M] \rrbracket) & & \llbracket !\Delta \rrbracket \otimes F(\llbracket [\text{out}(Q)] \rrbracket) \\ \downarrow F(f_M) & & \downarrow \eta' \\ F^2[A_m] & & F(\llbracket !\Delta \rrbracket \otimes \llbracket [\text{out}(Q)] \rrbracket) \\ & & \downarrow F(f_N) \\ & & F^2[A_n] \\ & \xrightarrow{\mu} & F[A] \xleftarrow{\mu} F^2[A_n] \end{array}$$

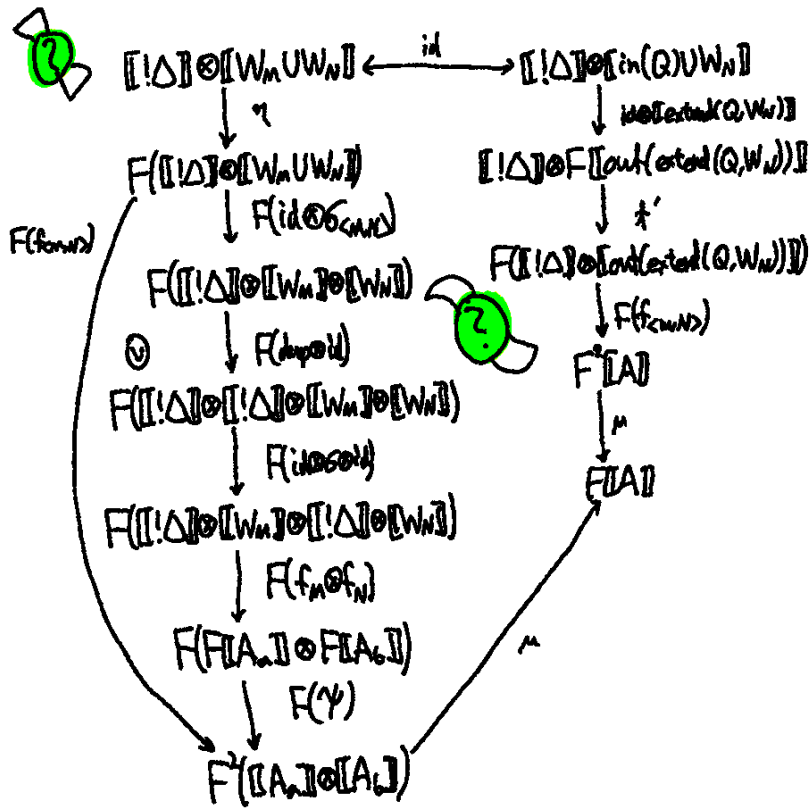
$$f_M = \llbracket \nu \text{Bind}(!\Delta, W_M, M, A_m) \rrbracket$$

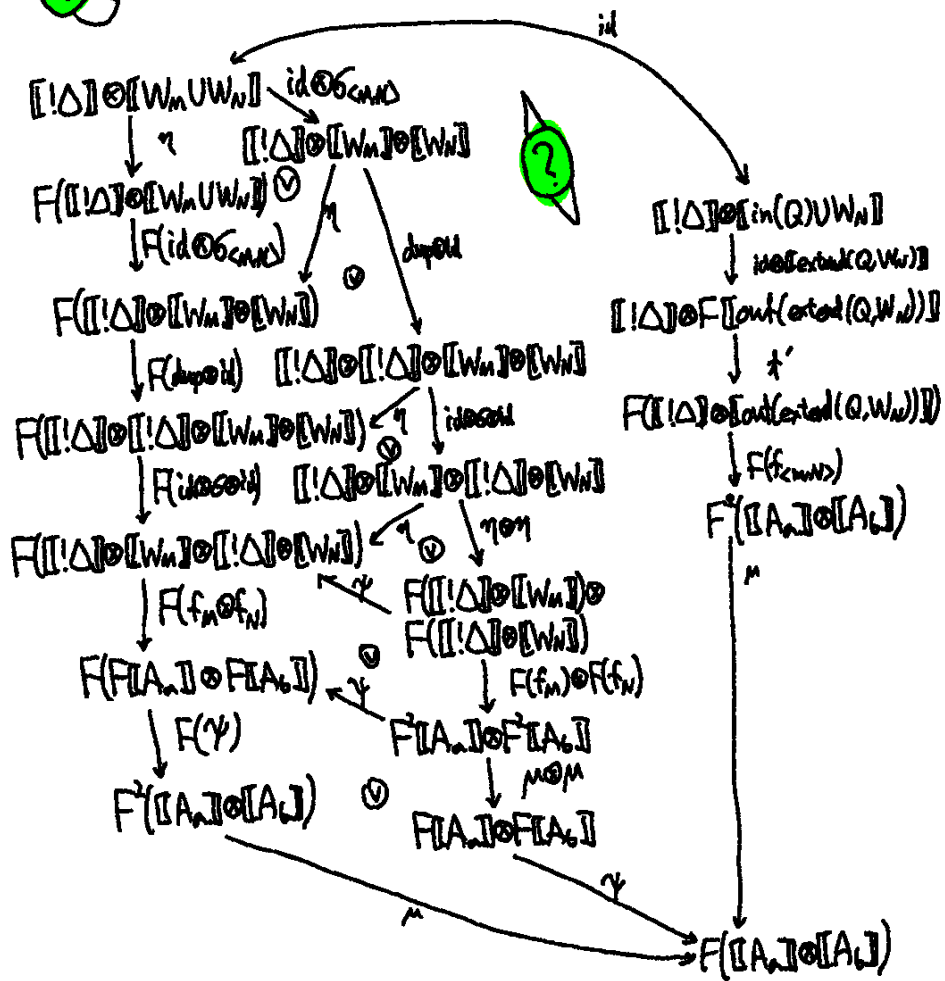
$$f_N = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q), m, A_n) \rrbracket$$

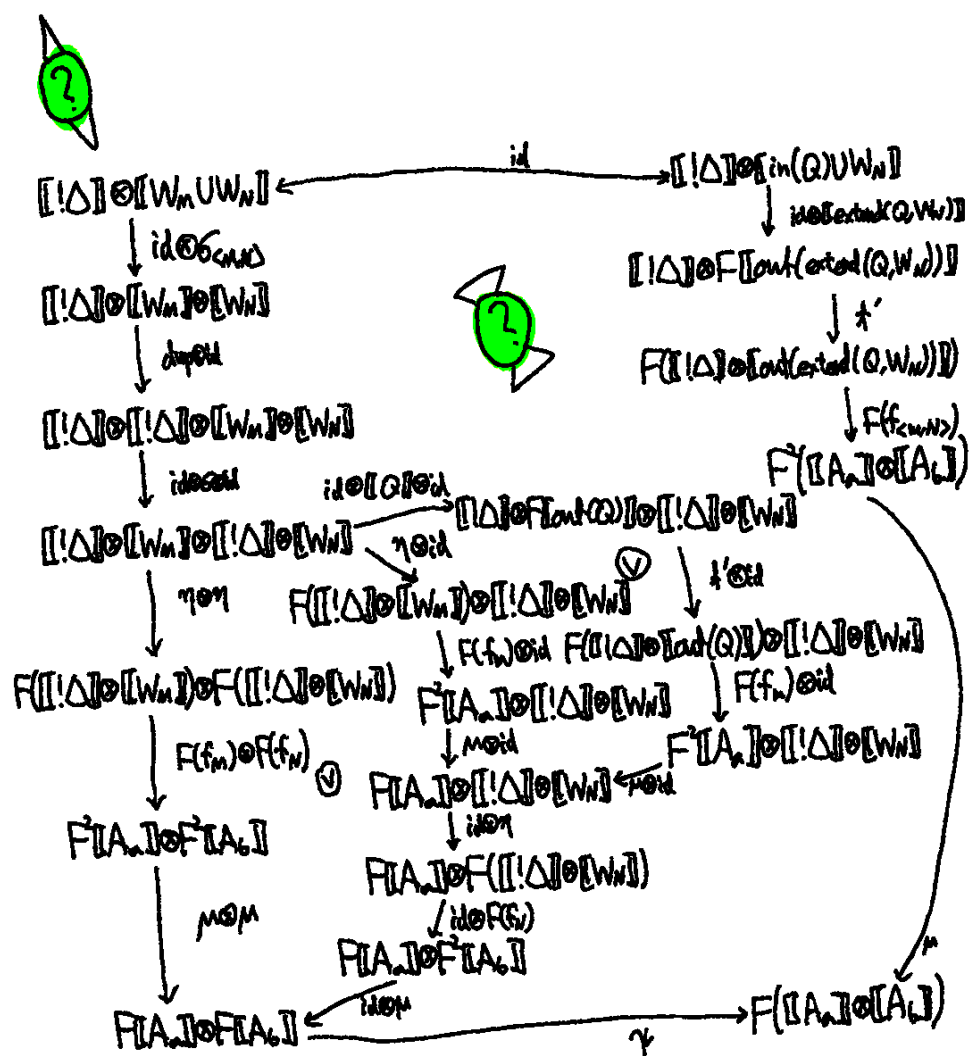
We show the following:

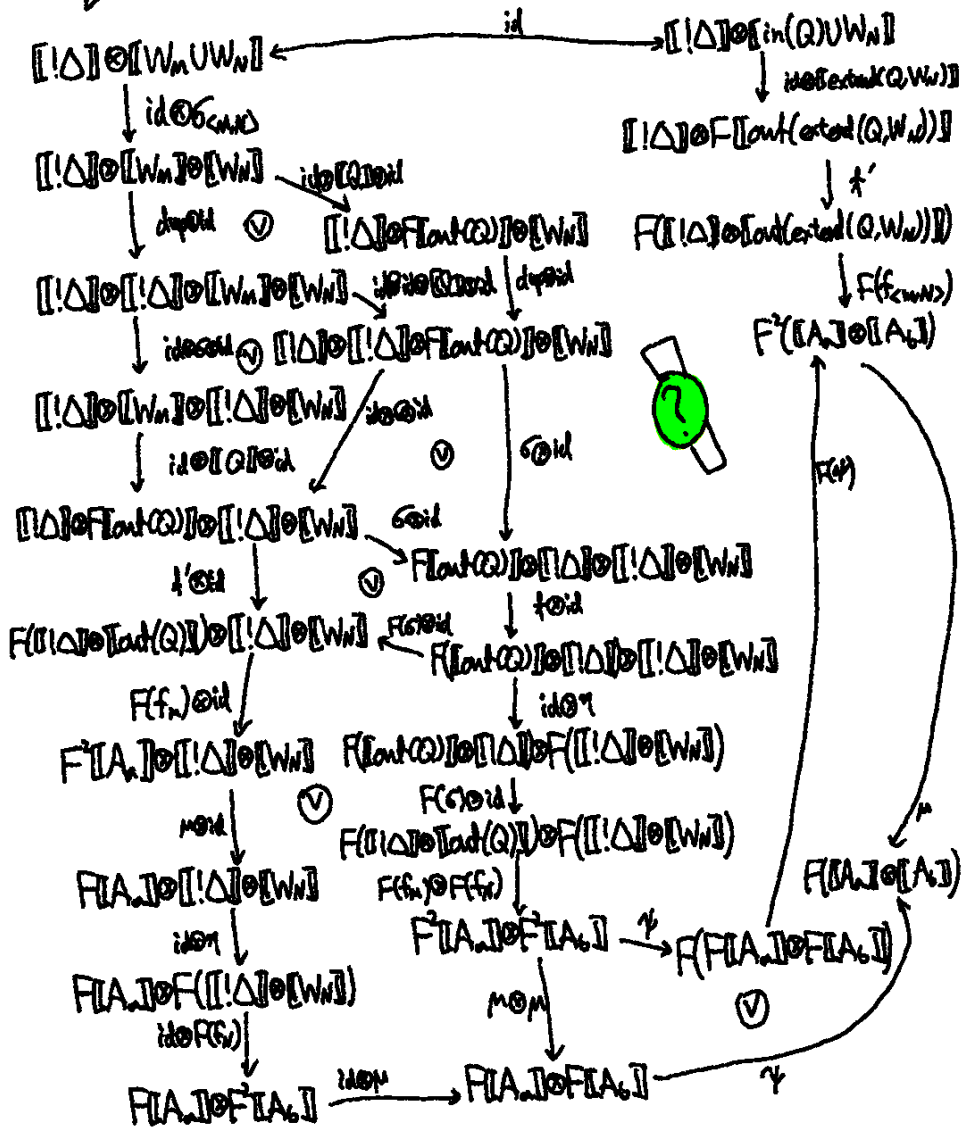


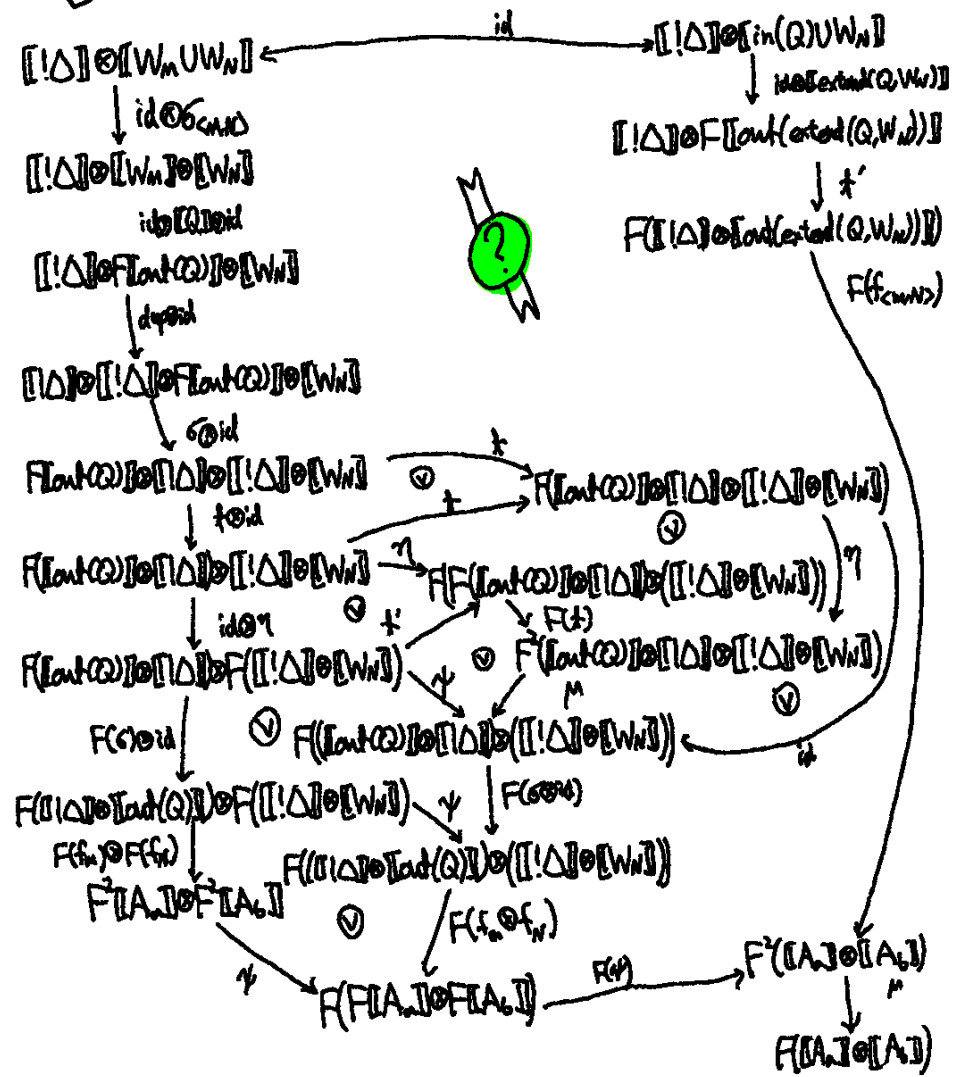
$$(f_n = [\nu \text{Bind}(! \Delta, W_n, N, A_b)], A = A_a \otimes A_b)$$




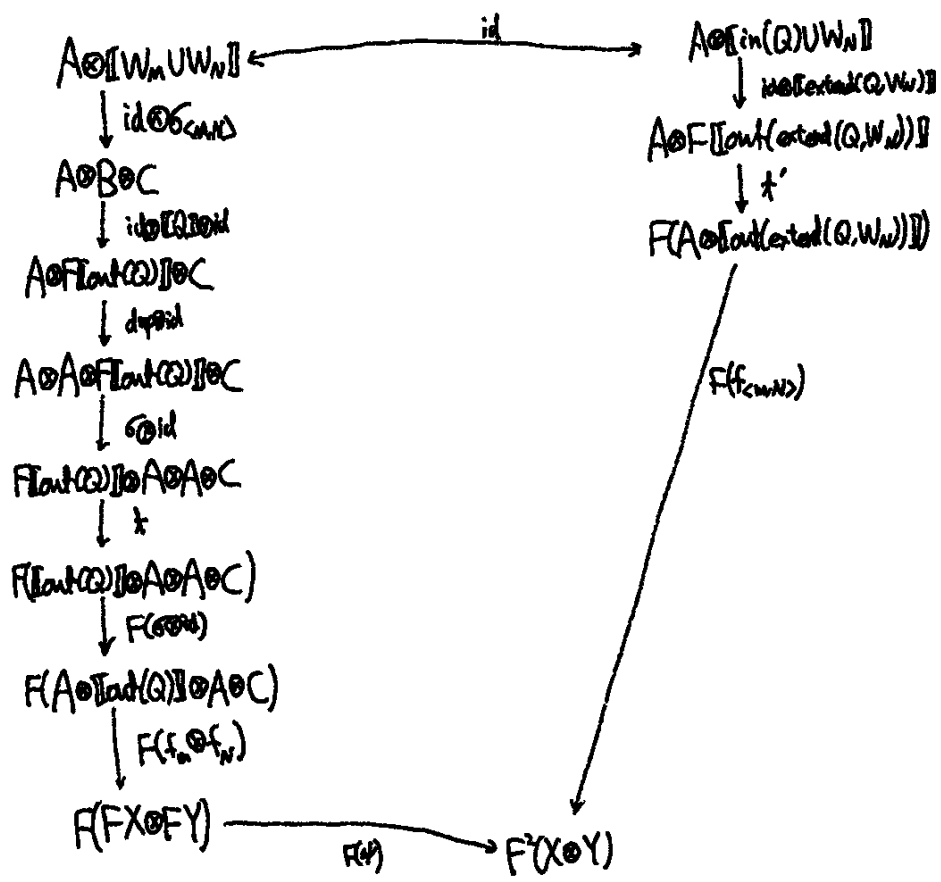








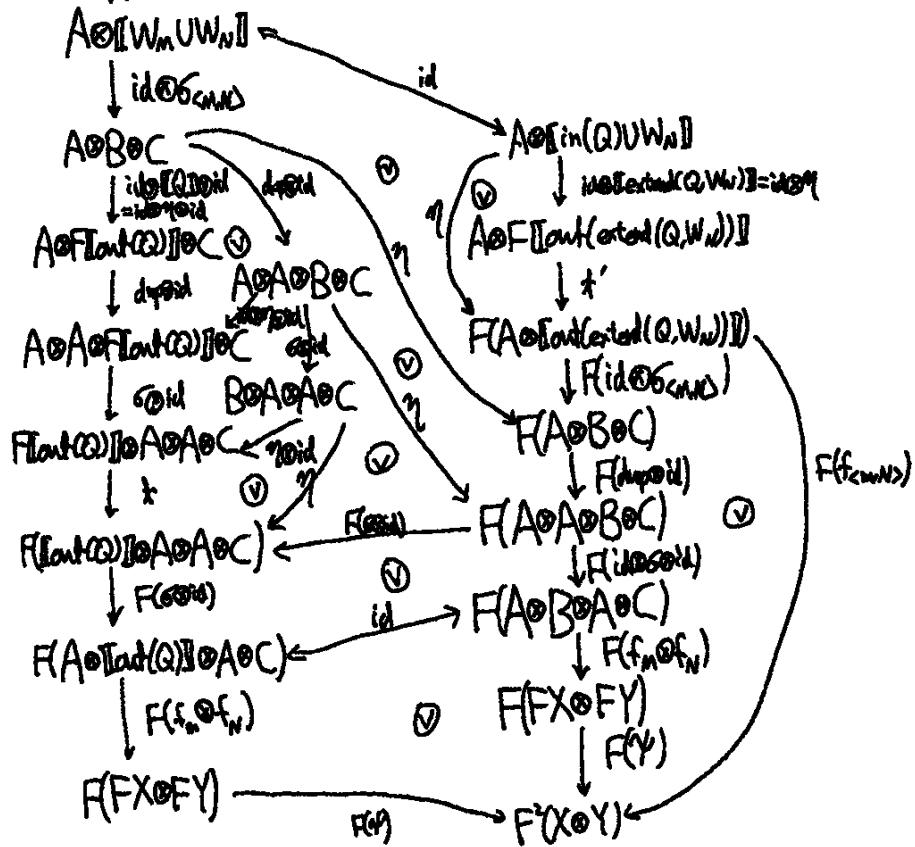
 Proof by induction



$$\textcircled{1} Q = \varepsilon(X)$$

$$\text{in}(Q) = W_N = \text{out}(Q)$$

$$\text{extend}(Q, W_N) = \varepsilon(XUW_N)$$

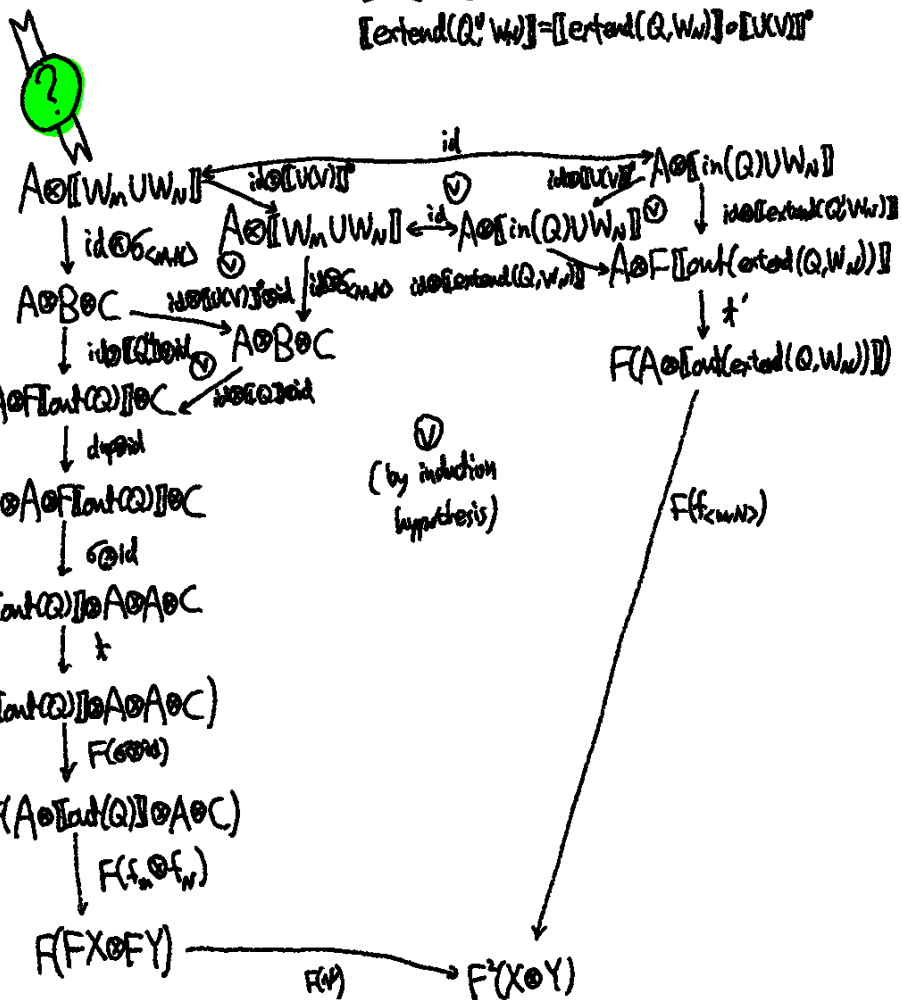


$$\textcircled{2} Q' = U(V)Q$$

$$\begin{aligned} \text{in}(Q') &= \text{in}(Q) \\ \text{out}(Q') &= \text{out}(Q) \end{aligned}$$

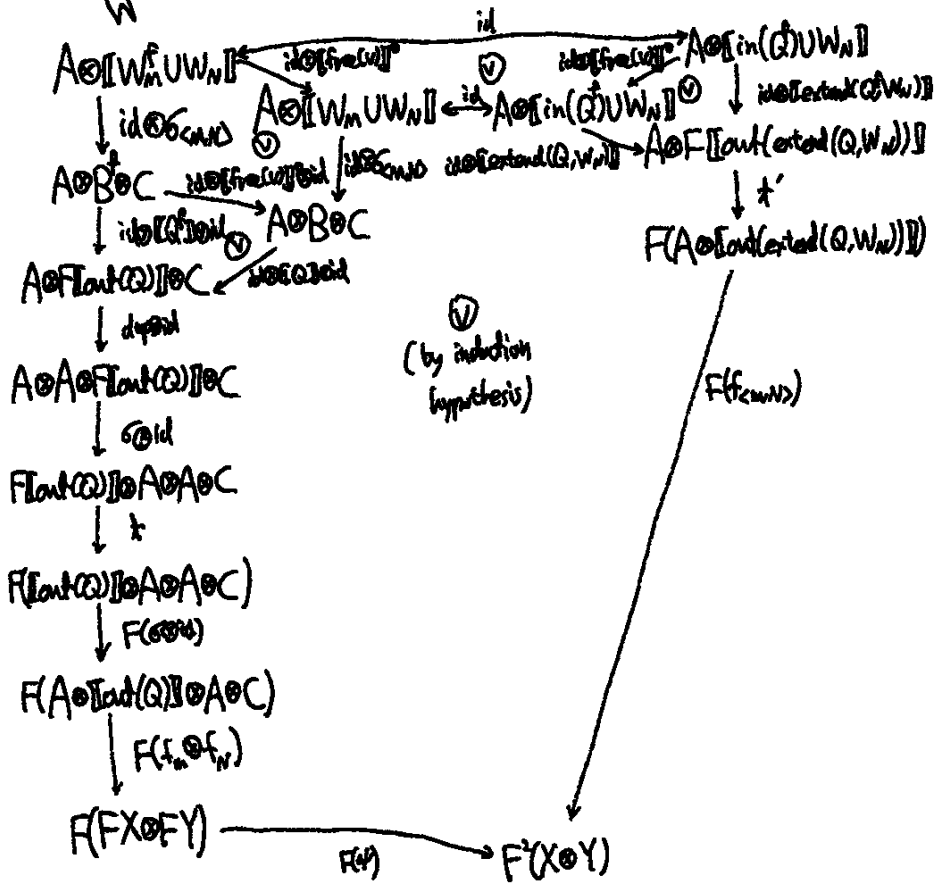
$$[Q'] = [Q] \cdot [U(V)]^*$$

$$[\text{extend}(Q', w)] = [\text{extend}(Q, w)] \cdot [U(V)]^*$$



$$\textcircled{3} Q^f = \text{free} \vee Q$$

$$\begin{aligned} \text{out}(Q^f) &= \text{out}(Q) \\ \text{out}(\text{extend}(Q^f, W)) &= \text{out}(\text{extend}(Q, W)) \\ \llbracket Q^f \rrbracket &= \llbracket Q \rrbracket \circ \llbracket \text{free}(W) \rrbracket \\ \llbracket \text{extend}(Q^f, W) \rrbracket &= \llbracket \text{extend}(Q, W) \rrbracket \circ \llbracket \text{free}(W) \rrbracket \end{aligned}$$



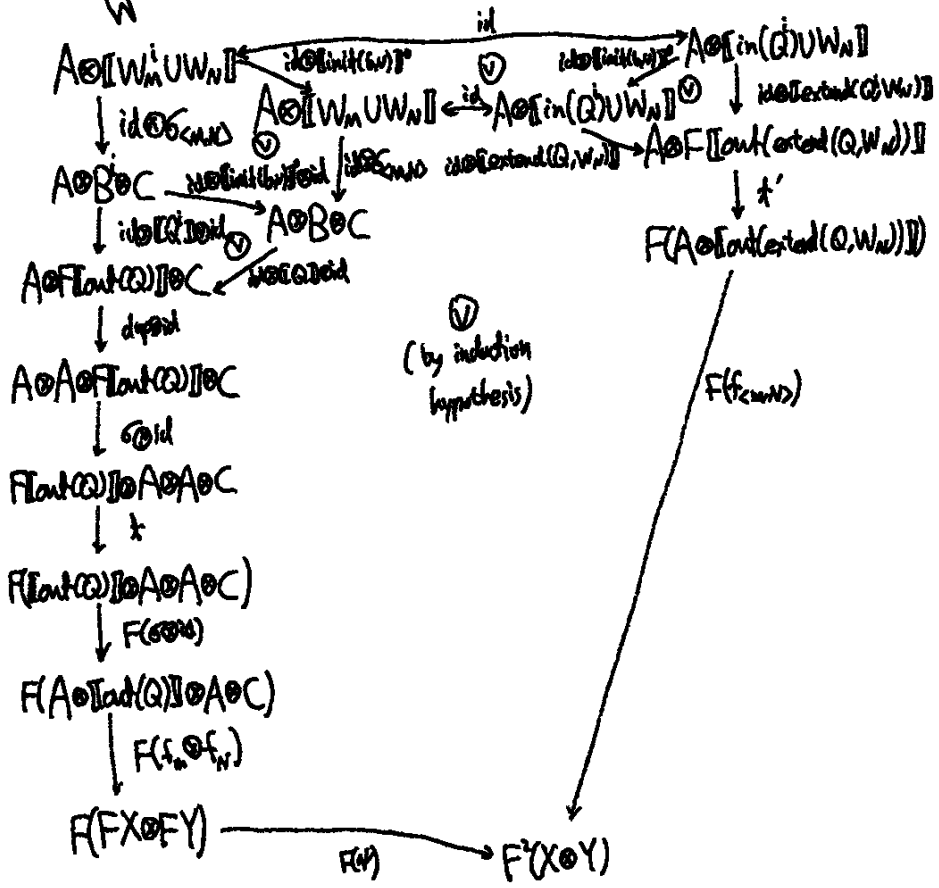
$$\textcircled{4} Q^i = \text{init } b \vee Q$$

$$\text{out}(Q^i) = \text{out}(Q)$$

$$\text{out}(\text{extend}(Q^i, W, v)) = \text{out}(\text{extend}(Q, W, v))$$

$$\llbracket Q^i \rrbracket = \llbracket Q \rrbracket \circ \llbracket \text{init}(b, v) \rrbracket^*$$

$$\llbracket \text{extend}(Q^i, W, v) \rrbracket = \llbracket \text{extend}(Q, W, v) \rrbracket \circ \llbracket \text{init}(b, v) \rrbracket^*$$



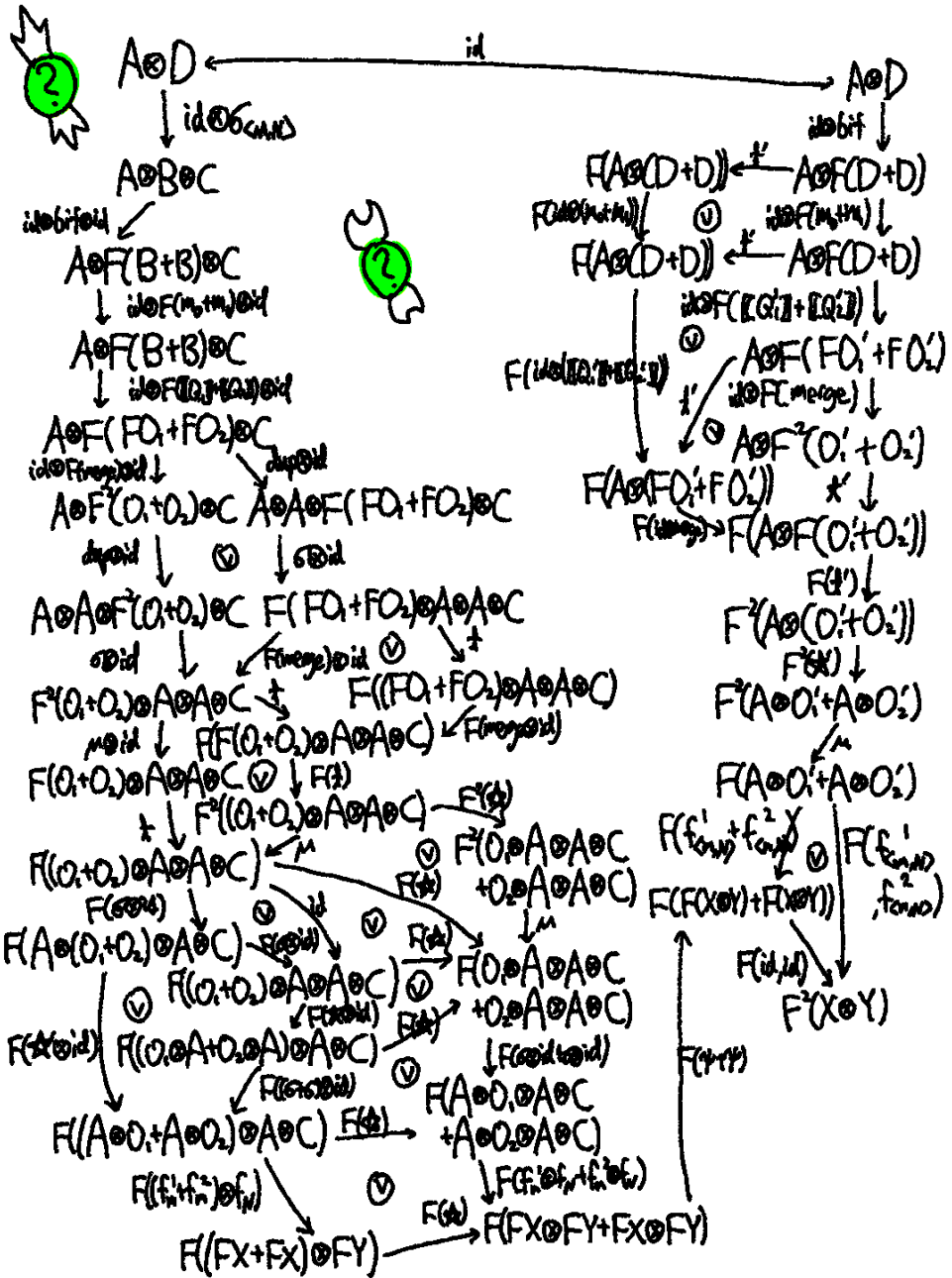
$$\begin{aligned}
 \textcircled{5} \quad Q^m &= \text{weav} \vee Q_1, Q_2 \\
 \text{out}(Q^m) &= [\text{out}(Q_1), \text{out}(Q_2)] & \text{in}(Q^m) &= \text{in}(Q_1) = \text{in}(Q_2) \\
 \text{out}(\text{extend}(Q^m, W_N)) &= [\text{out}(Q_1'), \text{out}(Q_2')] & \text{in}(Q^{m'}) &= \text{in}(Q^m) \cup W_N \\
 (Q^m)' &= \text{extend}(Q^m, W_N) = \text{weav} \vee Q_1', Q_2' \\
 (Q_1' = \text{extend}(Q_1, W_N) \quad Q_2' = \text{extend}(Q_2, W_N))
 \end{aligned}$$

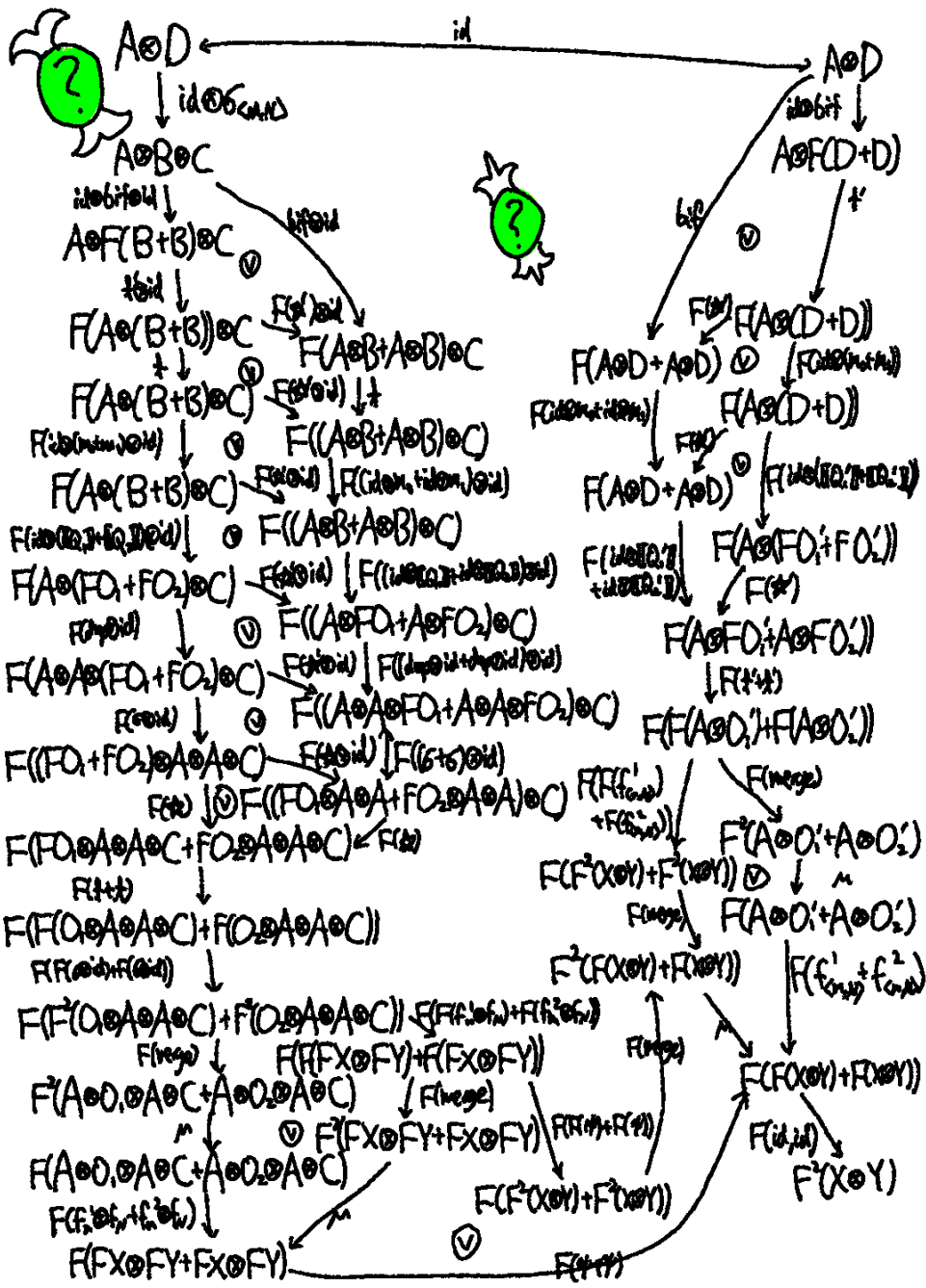
$$\begin{aligned}
 [Q^m] &= [[\text{in}(\text{weav} \vee Q_1, Q_2)]] \\
 &\downarrow \text{bit} \\
 &F([\text{in}(\text{weav} \vee Q_1, Q_2)] \\
 &\quad + [\text{in}(\text{weav} \vee Q_1, Q_2)]) \\
 &\downarrow F([Q_1] \circ [\text{weav}(u, v)]^* \\
 &\quad + [Q_2] \circ [\text{weav}(u, v)]^*) \\
 &F(F[\text{out}(Q_1)] + F[\text{out}(Q_2)]) \\
 &\downarrow \text{FC merge} \\
 &F^2([\text{out}(Q_1)] + [\text{out}(Q_2)]) \\
 &\downarrow M \\
 &F([\text{out}(Q_1)] + [\text{out}(Q_2)])
 \end{aligned}$$

$$\begin{aligned}
 f_m &= [[\Delta] \circ [\text{out}(Q_1)] + [\text{out}(Q_2)]] \\
 &\downarrow \star' \\
 &[[\Delta] \circ [\text{out}(Q_1)] + [[\Delta] \circ [\text{out}(Q_2)]] \\
 &\downarrow (f_m^1, f_m^2) \\
 &F[A_n]
 \end{aligned}$$

$$\begin{aligned}
 f_{m'} &= [[\Delta] \circ ([\text{out}(Q_1)] + [\text{out}(Q_2)])] \\
 &\downarrow \star' \\
 &[[\Delta] \circ [\text{out}(Q_1)] + [[\Delta] \circ [\text{out}(Q_2)]] \\
 &\downarrow (f_{m'}^1, f_{m'}^2) \\
 &F([A_n] \circ [A_b])
 \end{aligned}$$

$$\begin{aligned}
 [Q^{m'}] &= [[\text{in}(Q^{m'})]] \\
 &\downarrow \text{bit} \\
 &F([\text{in}(Q^{m'})] \\
 &\quad + [\text{in}(Q^{m'})]) \\
 &\downarrow F([Q_1'] \circ [\text{weav}(u, v)]^* \\
 &\quad + [Q_2'] \circ [\text{weav}(u, v)]^*) \\
 &F(F[\text{out}(Q_1')] + F[\text{out}(Q_2')]) \\
 &\downarrow \text{FC merge} \\
 &F^2([\text{out}(Q_1')] + [\text{out}(Q_2')]) \\
 &\downarrow M \\
 &F([\text{out}(Q_1')] + [\text{out}(Q_2')])
 \end{aligned}$$





A.10 . Congruence: on the right of a pair

$$\frac{(\mathcal{E}W_N, M) \rightarrow (Q, m) \quad \text{all}(Q) \cap W_N = \emptyset}{(\mathcal{E}(W_M \cup W_N), \langle N, M \rangle) \rightarrow (\text{extend}(Q, W_N), \langle N, m \rangle)}$$

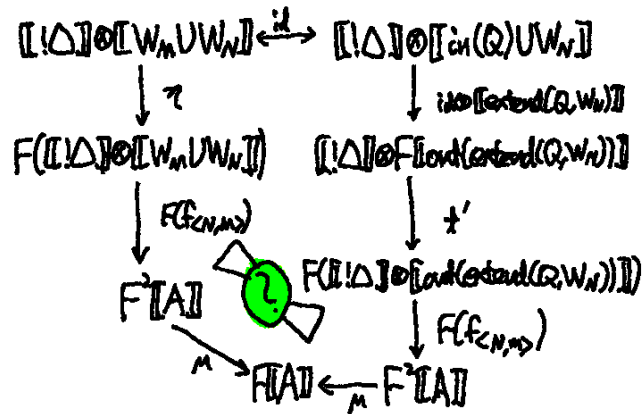
$$\begin{aligned} & \llbracket !\Delta \vdash (\mathcal{E}(W_M \cup W_N), \langle N, M \rangle) : A \rrbracket \\ &= \llbracket (\llbracket \mathcal{E}W_M \cup W_N \rrbracket), f_{\langle N, M \rangle} \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket W_M \cup W_N \rrbracket \xrightarrow{\eta} F(\llbracket !\Delta \rrbracket \otimes \llbracket W_M \cup W_N \rrbracket) \xrightarrow{F(f_{\langle N, M \rangle})} F^2[A] \xrightarrow{M} F[A] \\ & f_{\langle N, M \rangle} = \llbracket \nu \text{Bind}(!\Delta, W_M \cup W_N, \langle N, M \rangle, A) \rrbracket \end{aligned}$$

$$\begin{aligned} & \llbracket !\Delta \vdash (\text{extend}(Q, W_N), \langle N, m \rangle) : A \rrbracket \quad \text{in}(\text{extend}(Q, W_N)) = \text{in}(Q) \cup W_N \\ &= \llbracket (\llbracket \text{extend}(Q, W_N) \rrbracket), f_{\langle N, m \rangle} \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \cup W_N \rrbracket \xrightarrow{\text{id} \otimes \llbracket \text{extend}(Q, W_N) \rrbracket} \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(\text{extend}(Q, W_N)) \rrbracket \\ & \quad \downarrow \ddagger' \\ & \quad F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(\text{extend}(Q, W_N)) \rrbracket) \\ & \quad \downarrow F(f_{\langle N, m \rangle}) \\ & \quad F^2[A] \\ & \quad \downarrow M \\ & \quad F[A] \\ & f_{\langle N, m \rangle} = \llbracket \nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \langle N, m \rangle, A) \rrbracket \end{aligned}$$

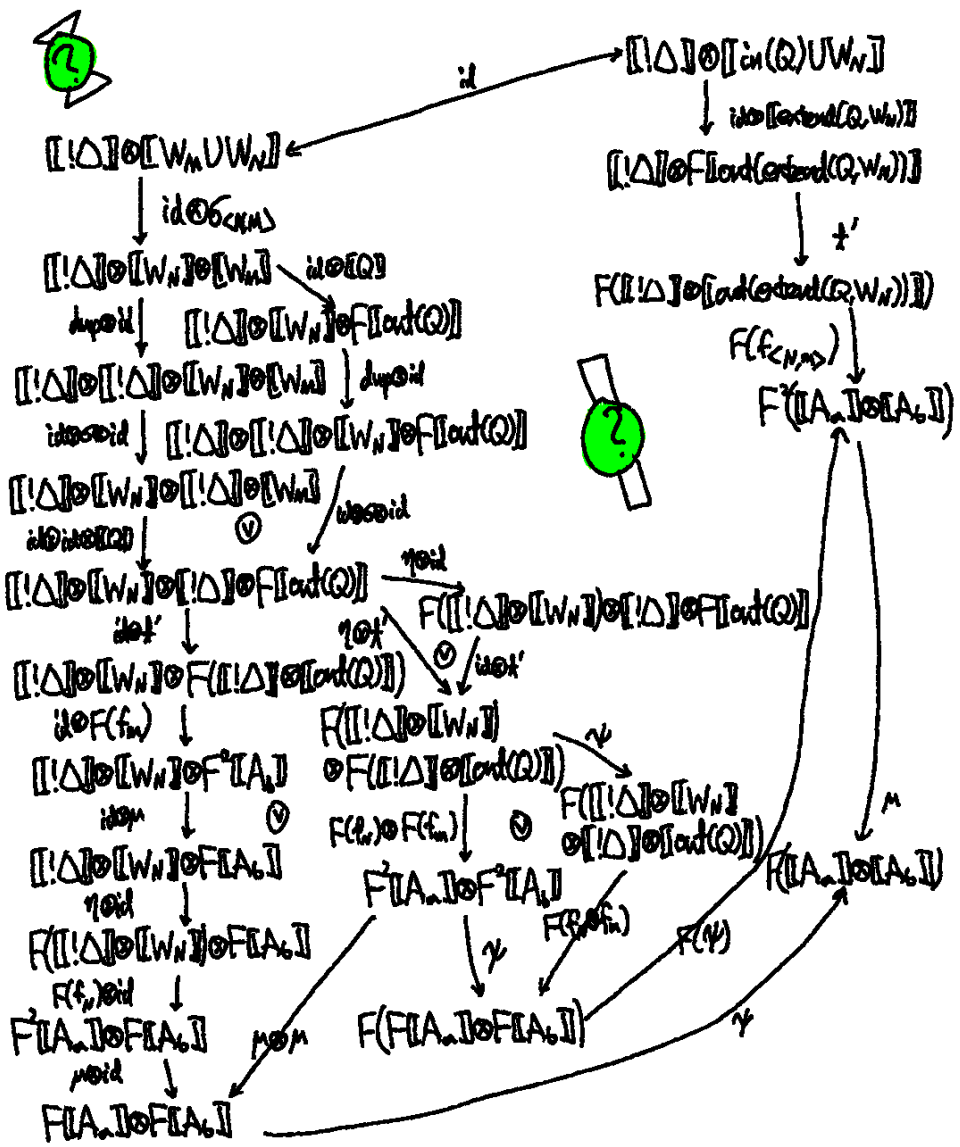
From the induction hypothesis:


$$\begin{array}{ccc} \llbracket !\Delta \rrbracket \otimes \llbracket W_M \rrbracket & \xleftarrow{\text{id}} & \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \\ \downarrow \eta & & \downarrow \text{id} \otimes \llbracket \text{in}(Q) \rrbracket \\ F(\llbracket !\Delta \rrbracket \otimes \llbracket W_M \rrbracket) & & \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{in}(Q) \rrbracket) \\ \downarrow F(f_M) & & \downarrow \ddagger' \\ F^2[A] & & F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \\ & & \downarrow F(f_M) \\ & & F^2[A] \\ & \xrightarrow{M} & F[A] \xleftarrow{M} F^2[A] \end{array} \quad \begin{array}{l} f_M = \llbracket \nu \text{Bind}(!\Delta, W_M, M, A) \rrbracket \\ f_m = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q), m, A) \rrbracket \end{array}$$

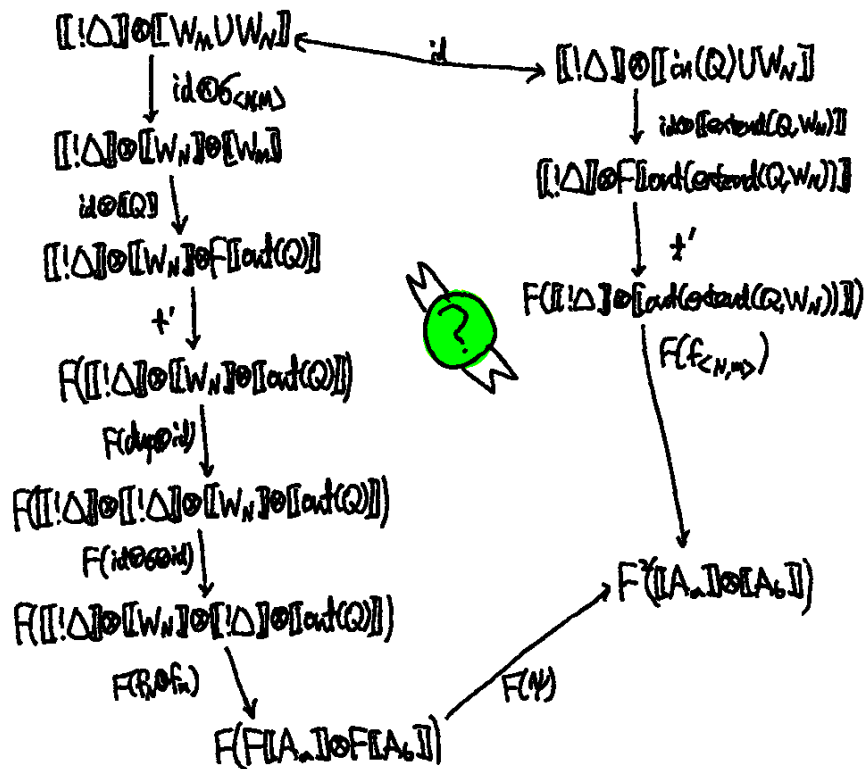
We show the following:



$$(f_N = [vBind(\Delta, W_n, N, A_n)]), A = A_n \otimes A_m$$



 Proof by induction on Q

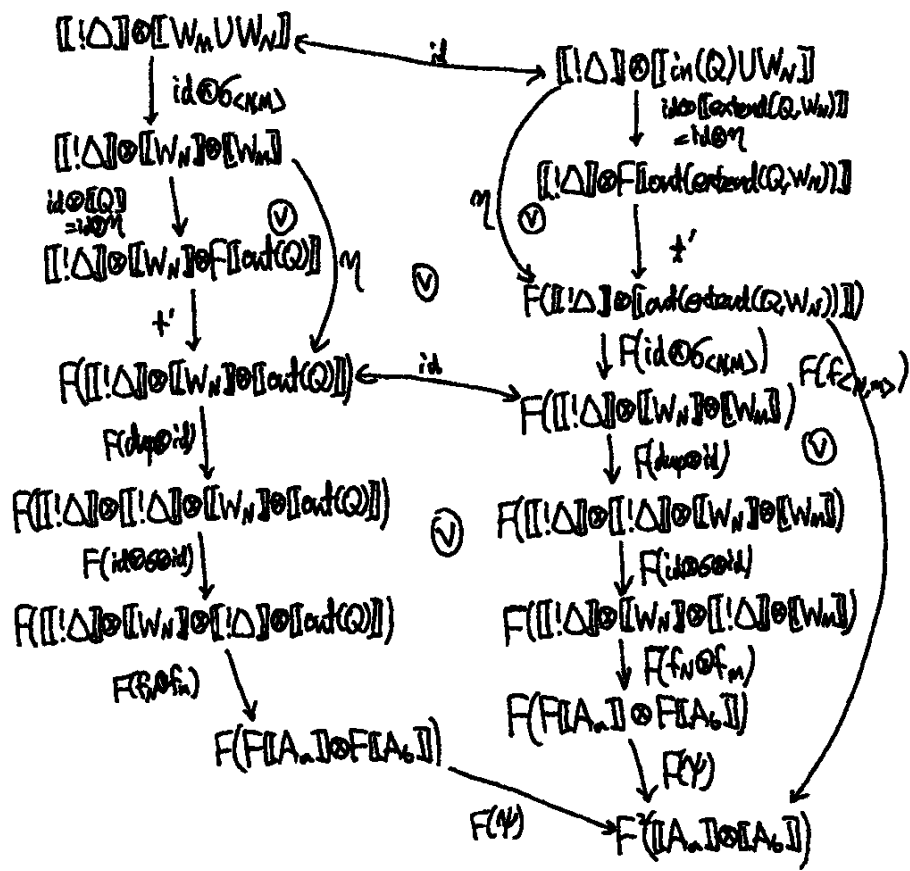


$$\textcircled{1} Q = \varepsilon(W_m) \quad \begin{array}{l} \text{in}(Q) = W_m \\ \text{out}(Q) = W_m \end{array}$$

$$\text{extend}(Q, W_N) = \varepsilon(W_m \cup W_N) \quad \text{out}(\text{extend}(Q, W_N)) = W_m \cup W_N$$

$$f_{\langle N, m \rangle} = [\text{bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \langle N, m \rangle, A)]$$

$$\begin{aligned} &= [!\Delta] \otimes [W_m \cup W_N] \\ &\quad \downarrow \text{id} \otimes \delta_{\langle N, m \rangle} \\ &= [!\Delta] \otimes [W_m] \otimes [W_N] \\ &\quad \downarrow \text{dup id} \\ &= [!\Delta] \otimes [!\Delta] \otimes [W_m] \otimes [W_N] \\ &\quad \downarrow \text{id} \otimes \text{id} \\ &= [!\Delta] \otimes [W_m] \otimes [!\Delta] \otimes [W_N] \\ &\quad \downarrow f_N \otimes f_m \\ &= F[A_m] \otimes F[A_N] \\ &\quad \downarrow \gamma \\ &= F([A_m] \otimes [A_N]) \end{aligned}$$



$$\textcircled{2} Q^u = U(V) Q$$

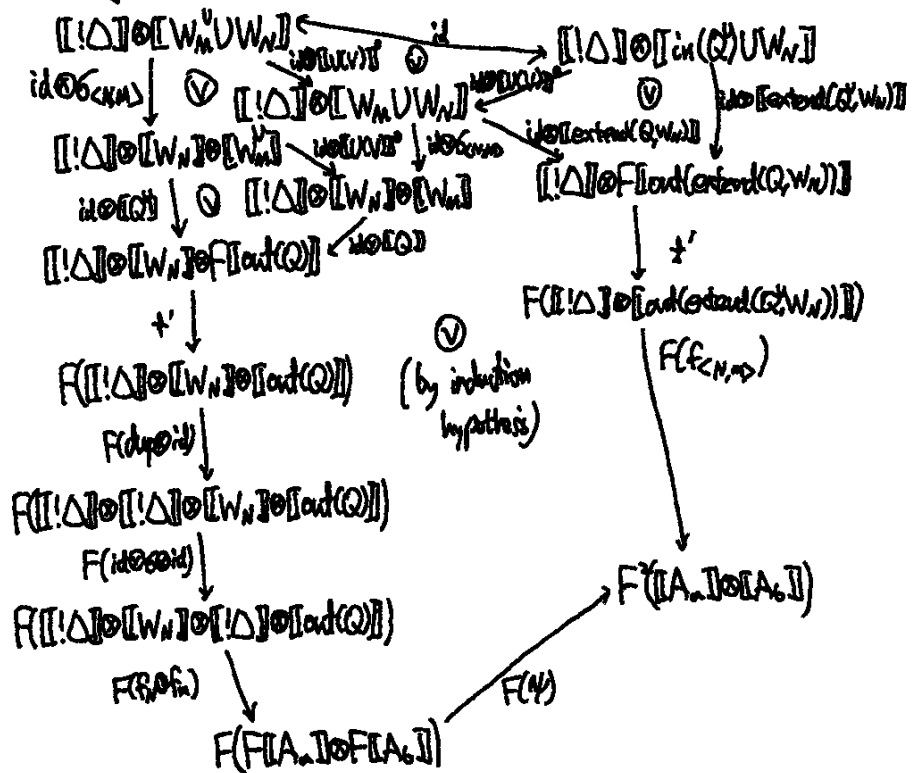
$$\text{extend}(Q^u, W_N) = U(V) \text{extend}(Q, W_N)$$

$$\text{out}(\text{extend}(Q^u, W_N)) = \text{out}(\text{extend}(Q, W_N))$$

$$\text{out}(Q^u) = \text{out}(Q)$$

$$\text{in}(Q^u) = W_M^u$$

$$\text{in}(\text{extend}(Q^u, W_N)) = W_M^u U W_N$$



③ $Q^f = \text{free } v \ Q$

$$\text{extend}(Q^f, W_N) = \text{free } v \ \text{extend}(Q, W_N)$$

$$\text{out}(\text{extend}(Q^f, W_N)) = \text{out}(\text{extend}(Q, W_N))$$

$$\text{out}(Q^f) = \text{out}(Q)$$

$$\text{in}(Q^f) = W_N^f$$

$$\text{in}(\text{extend}(Q^f, W_N)) = W_N^f \cup W_N$$



$$\textcircled{4} Q^i = \text{init } b \vee Q$$

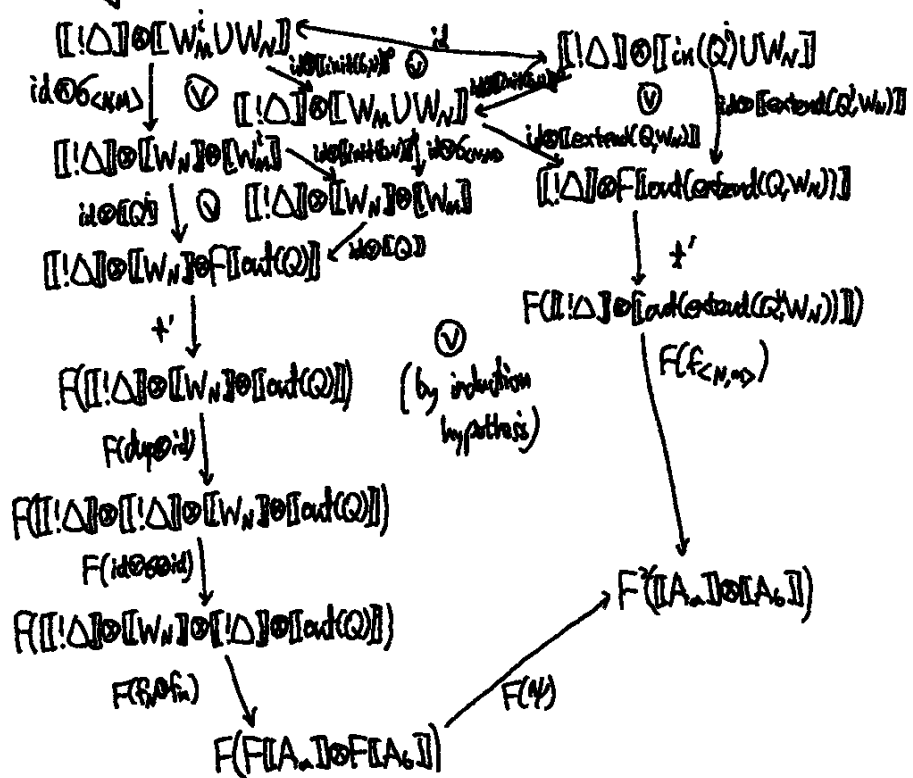
$$\text{extend}(Q^i, W_N) = \text{init } b \vee \text{extend}(Q, W_N)$$

$$\text{out}(\text{extend}(Q^i, W_N)) = \text{out}(\text{extend}(Q, W_N))$$

$$\text{out}(Q^i) = \text{out}(Q)$$

$$\text{in}(Q^i) = W_M^i$$

$$\text{in}(\text{extend}(Q^i, W_N)) = W_M^i \cup W_N$$



$$\textcircled{5} Q^m = \text{menc} \vee Q_1 Q_2$$

$$(Q^m)' = \text{extend}(Q^m, W_N) = \text{menc} \vee Q_1' Q_2'$$

$$\text{out}((Q^m)') = [\text{out}(Q_1'), \text{out}(Q_2')] \quad \text{in}(Q^m) = W_N^m = \text{in}(Q_1) = \text{in}(Q_2)$$

$$\text{out}(Q^m) = [\text{out}(Q_1), \text{out}(Q_2)] \quad \text{in}((Q^m)') = W_N^m \vee W_N = \text{in}(Q_1') = \text{in}(Q_2')$$

$$Q_1' = \text{extend}(Q_1, W_N), \quad Q_2' = \text{extend}(Q_2, W_N)$$

$$[Q^m] = [in(\text{menc} \vee Q_1 Q_2)]$$

$$\begin{aligned} &\downarrow \text{bit} \\ &F([in(\text{menc} \vee Q_1 Q_2)] \\ &\quad + [in(\text{menc} \vee Q_1 Q_2)]) \\ &\quad \downarrow F([Q_1] \circ [menc(u,0)]^* \\ &\quad \quad + [Q_2] \circ [menc(u,1)]^*) \\ &F(F[in(Q_1)] + F[out(Q_2)]) \\ &\quad \downarrow \text{FC merge} \\ &F^2([in(Q_1)] + [out(Q_2)]) \\ &\quad \downarrow M \\ &F([in(Q_1)] + [out(Q_2)]) \end{aligned}$$

$$[Q^m]' = [in(Q^m)']$$

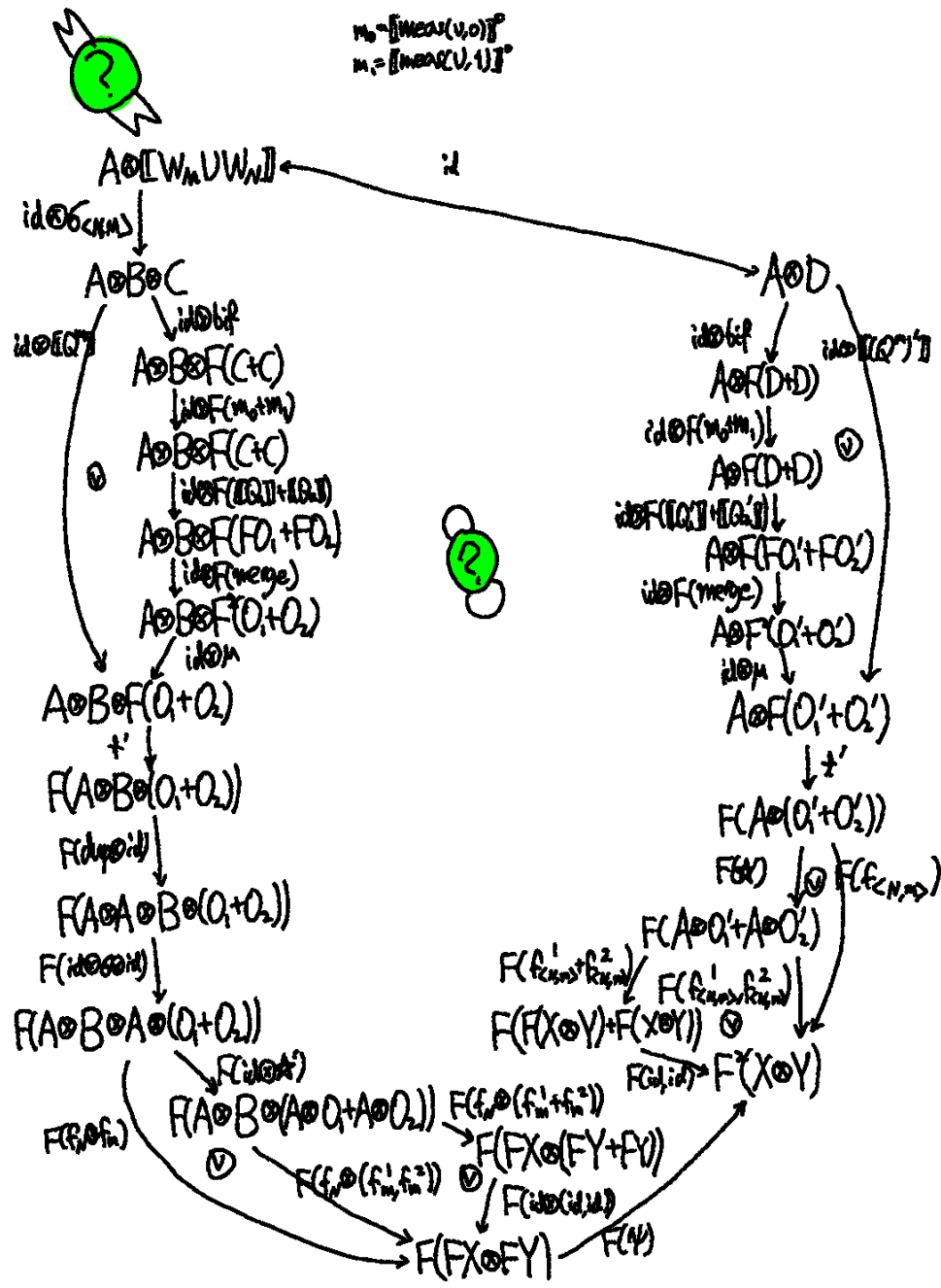
$$\begin{aligned} &\downarrow \text{bit} \\ &F([in(Q^m)'] \\ &\quad + [in(Q^m)']) \\ &\quad \downarrow F([Q_1] \circ [menc(u,0)]^* \\ &\quad \quad + [Q_2] \circ [menc(u,1)]^*) \\ &F(F[in(Q_1)'] + F[out(Q_2)']) \\ &\quad \downarrow \text{FC merge} \\ &F^2([in(Q_1)'] + [out(Q_2)']) \\ &\quad \downarrow M \\ &F([in(Q_1)'] + [out(Q_2)']) \end{aligned}$$

$$f_m = [! \Delta] \circ ([out(Q_1)] + [out(Q_2)])$$

$$\begin{aligned} &\downarrow \star' \\ &[! \Delta] \circ [out(Q_1)] + [! \Delta] \circ [out(Q_2)] \\ &\quad \downarrow (f_m^1, f_m^2) \\ &F[A_m] \end{aligned}$$

$$f_{m'} = [! \Delta] \circ ([out(Q_1)'] + [out(Q_2)'])$$

$$\begin{aligned} &\downarrow \star' \\ &[! \Delta] \circ [out(Q_1)'] + [! \Delta] \circ [out(Q_2)'] \\ &\quad \downarrow (f_{m'}^1, f_{m'}^2) \\ &F[A_m'] \circ [A_m] \end{aligned}$$



A.11 . Congruence: if-then-else

$$\frac{(\mathcal{E}(W_M), M) \rightarrow (Q, m) \quad \text{all}(Q) \cap W_N = \emptyset}{(\mathcal{E}(W_M \cup W_N), \text{if } M \text{ then } M_a \text{ else } M_b) \rightarrow (\text{extend}(Q, W_N), \text{if } m \text{ then } M_a \text{ else } M_b)}$$

$$\begin{aligned} & \llbracket !\Delta \vdash (\mathcal{E}(W_M \cup W_N), \text{if } M \text{ then } M_a \text{ else } M_b) : A \rrbracket \\ &= \llbracket (\llbracket \mathcal{E}(W_M \cup W_N) \rrbracket, f) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket W_M \cup W_N \rrbracket \xrightarrow{\eta} F(\llbracket !\Delta \rrbracket \otimes \llbracket W_M \cup W_N \rrbracket) \xrightarrow{Fg} F^2[A] \xrightarrow{\mu} F[A] \\ & f = \llbracket \nu \text{Bind}(!\Delta, W_M \cup W_N, \text{if } M \text{ then } M_a \text{ else } M_b, A) \rrbracket \\ & \llbracket !\Delta \vdash (\text{extend}(Q, W_N), \text{if } m \text{ then } M_a \text{ else } M_b) : A \rrbracket \\ &= \llbracket (\llbracket \text{extend}(Q, W_N) \rrbracket, g) \rrbracket \quad \text{in}(\text{extend}(Q, W_N)) = \text{in}(Q) \cup W_N \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \cup W_N \rrbracket \xrightarrow{\text{in}(\text{extend}(Q, W_N))} \llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(\text{extend}(Q, W_N)) \rrbracket \\ & \quad \downarrow \eta' \\ & \quad F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(\text{extend}(Q, W_N)) \rrbracket) \\ & \quad \downarrow Fg \\ & \quad F^2[A] \\ & \quad \downarrow \mu \\ & \quad F[A] \end{aligned}$$

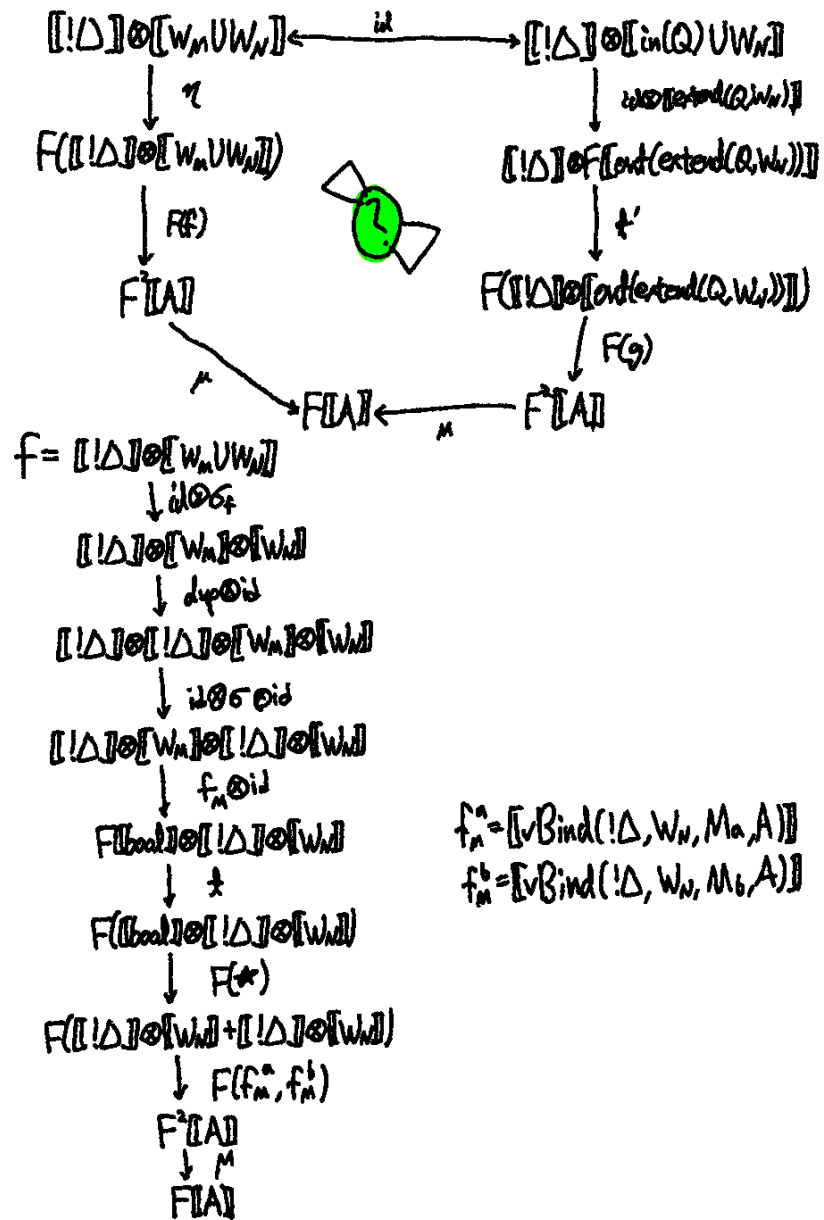
$$g = \llbracket \nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \text{if } m \text{ then } M_a \text{ else } M_b, A) \rrbracket$$

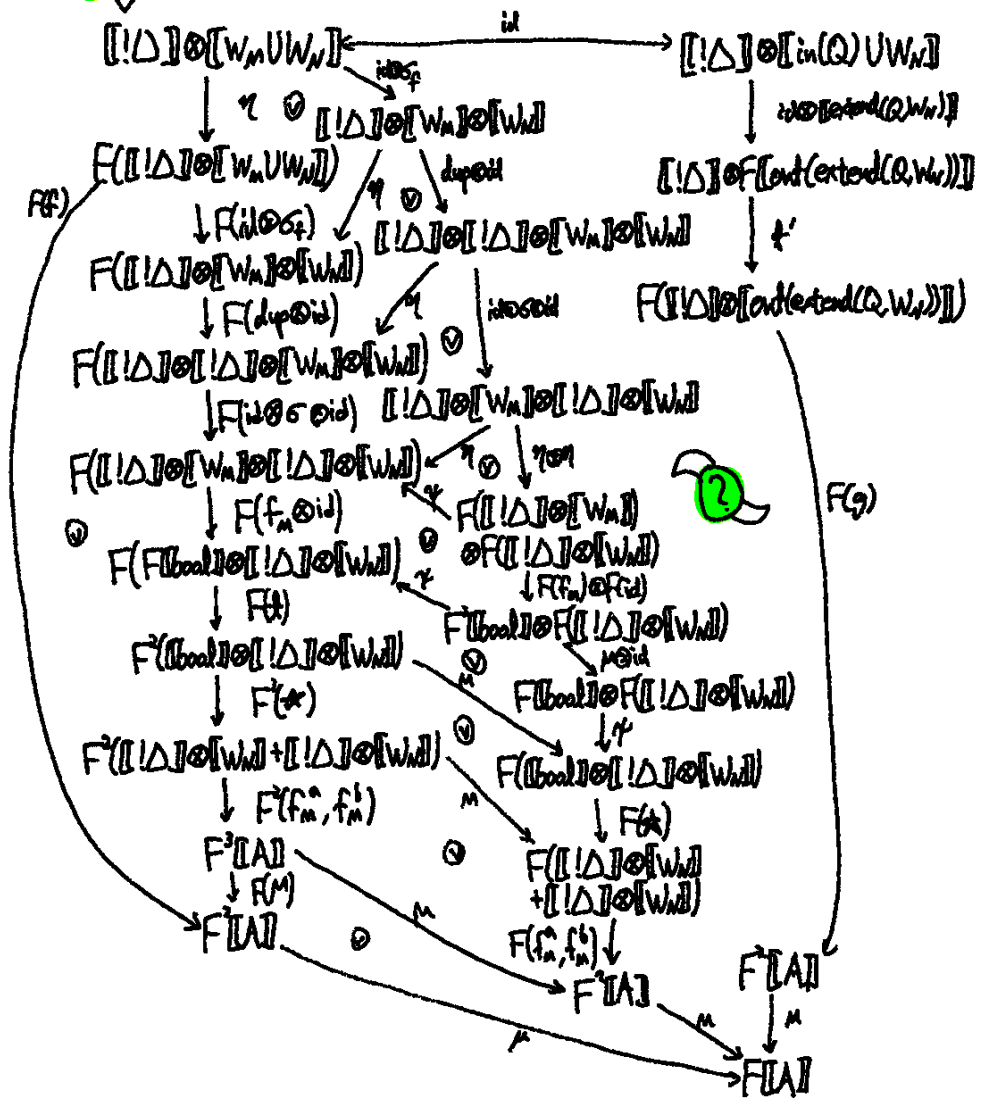
From the induction hypothesis:

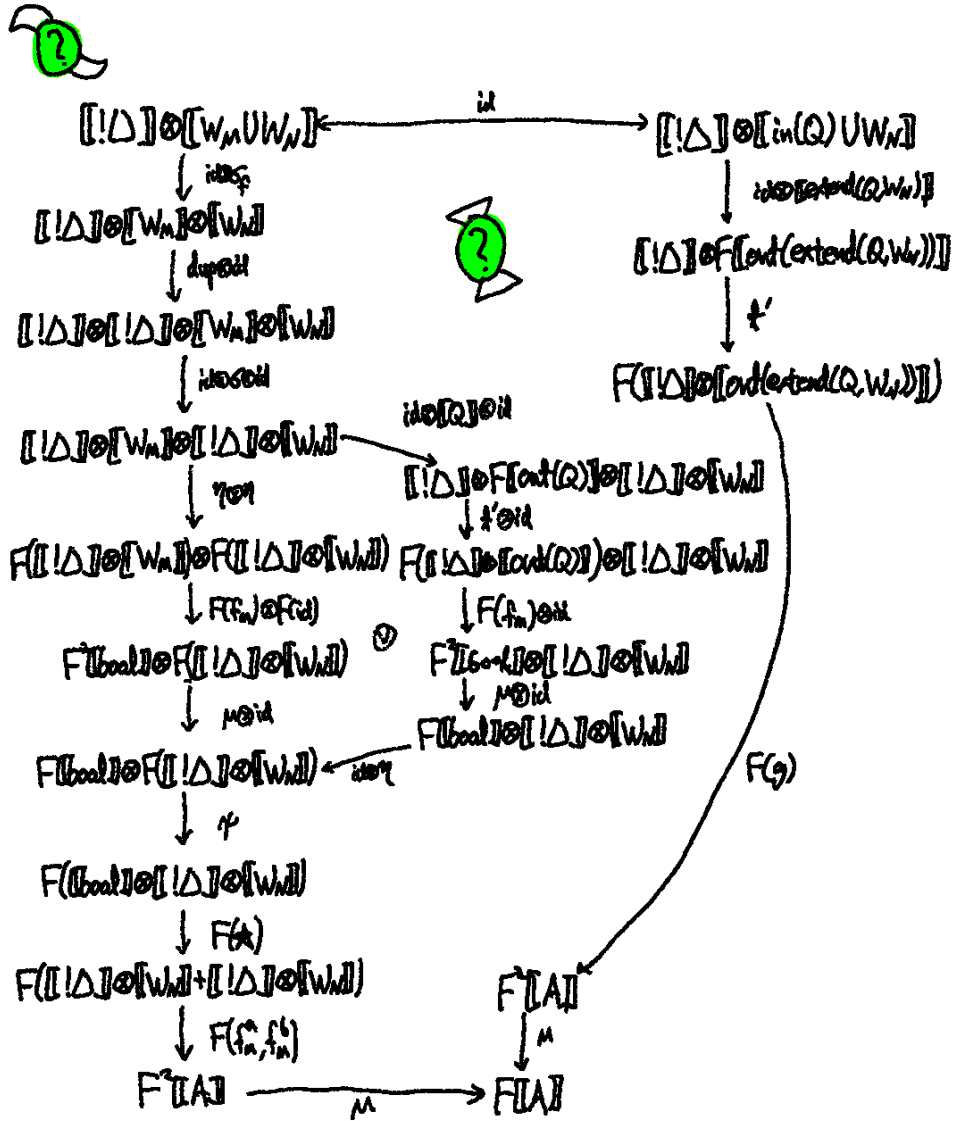
$$\begin{array}{ccc} \llbracket !\Delta \rrbracket \otimes \llbracket W_M \rrbracket & \xleftarrow{\eta} & \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \\ \downarrow \eta & & \downarrow \eta \otimes \text{id} \\ F(\llbracket !\Delta \rrbracket \otimes \llbracket W_M \rrbracket) & & \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{in}(Q) \rrbracket) \\ \downarrow Ff_m & & \downarrow \eta' \\ F^2[\llbracket \text{in}(Q) \rrbracket] & \xleftarrow{\mu} & F^2[\llbracket \text{out}(Q) \rrbracket] \\ \downarrow \mu & & \downarrow \mu \\ F[\llbracket \text{in}(Q) \rrbracket] & \xleftarrow{\mu} & F[\llbracket \text{out}(Q) \rrbracket] \end{array}$$

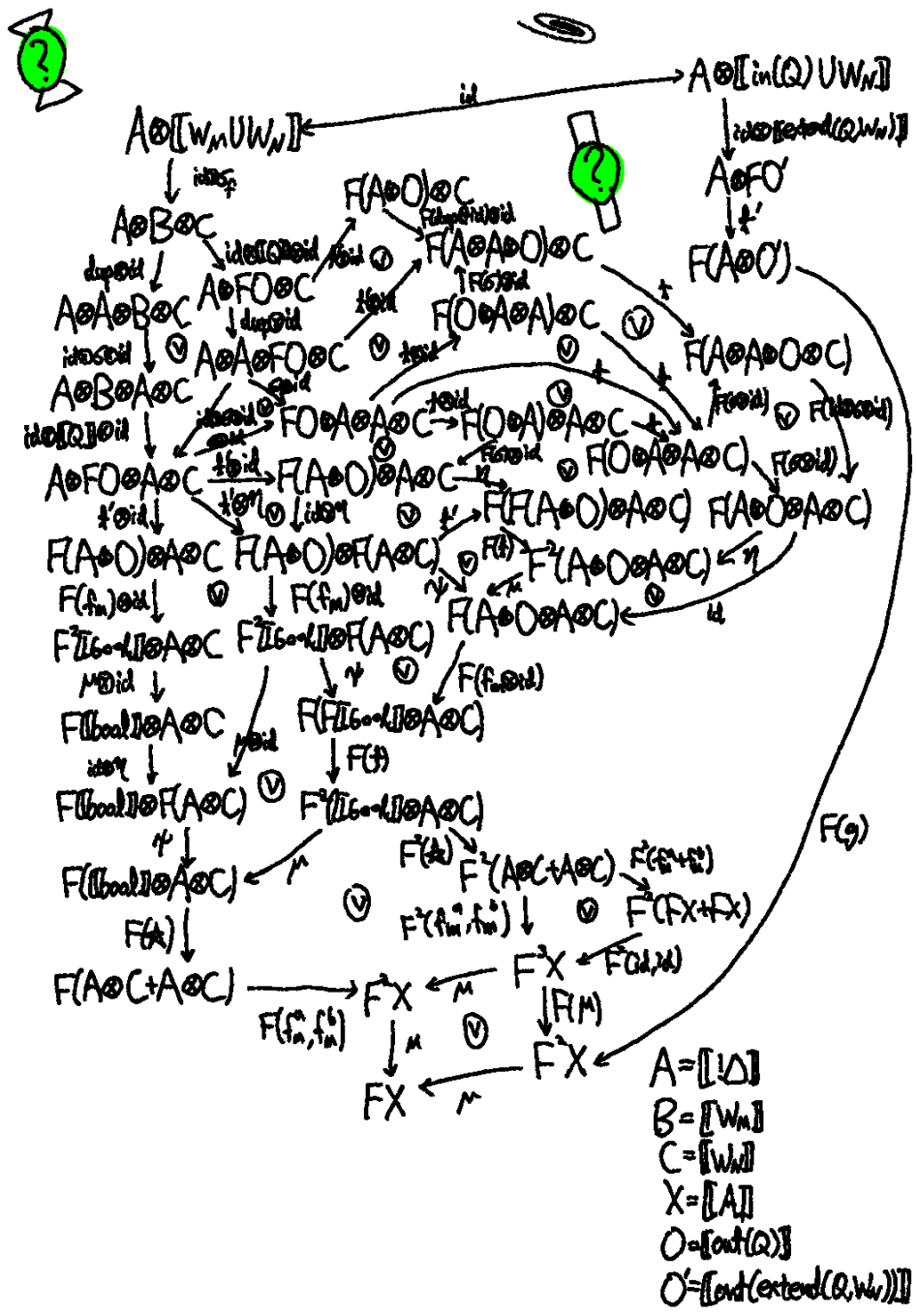
$f_m = \llbracket \nu \text{Bind}(!\Delta, W_M, m, \text{in}(Q)) \rrbracket$
 $f_n = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q), m, \text{out}(Q)) \rrbracket$

We show the following:





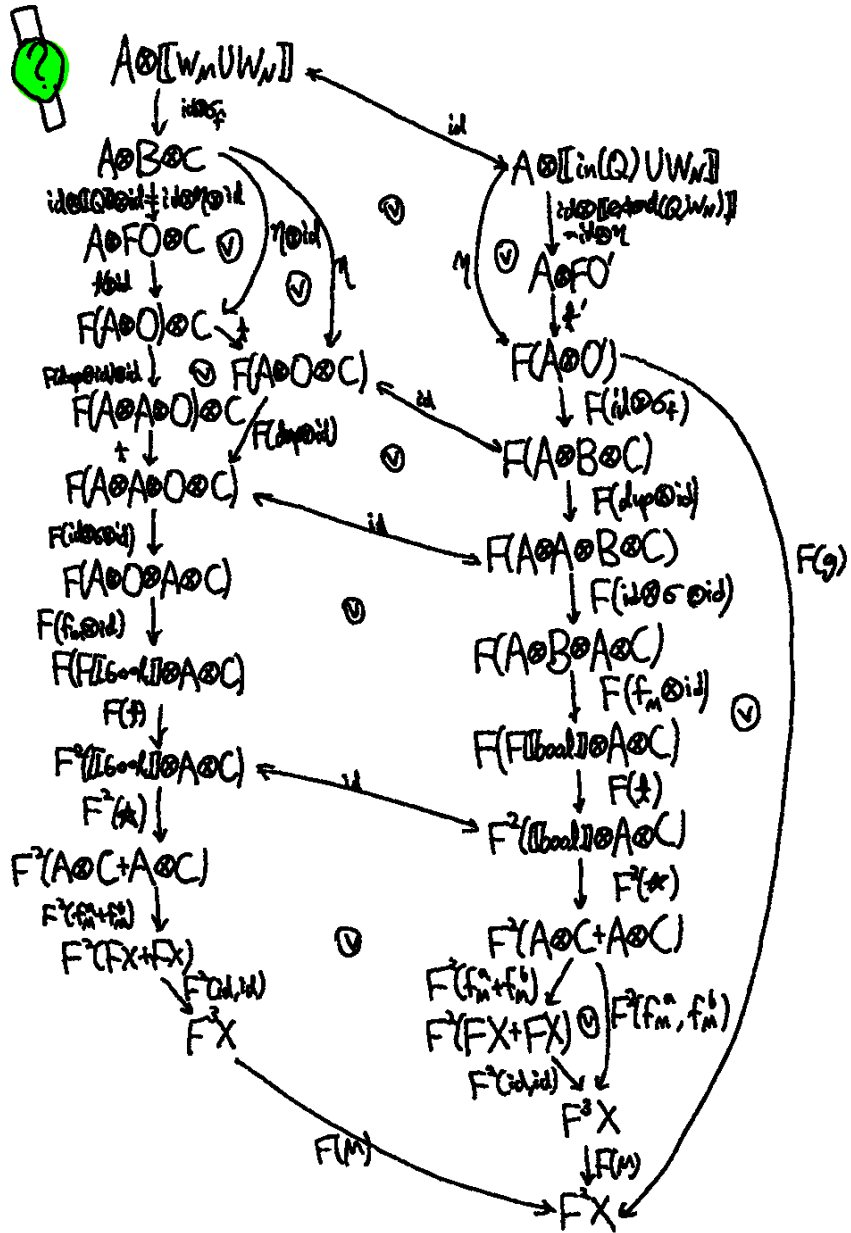




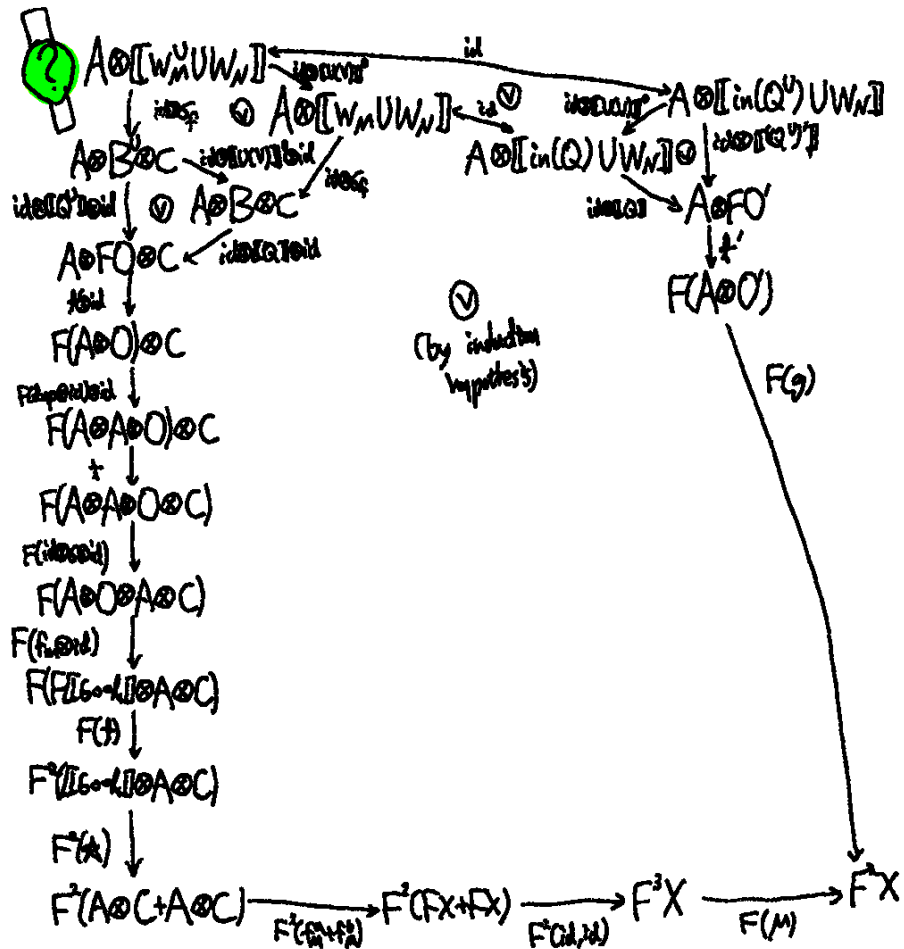
② Proof by induction on Q

$$\begin{aligned} \textcircled{1} \quad Q &= \varepsilon(W_M) \\ \text{extend}(Q, W_N) &= \varepsilon(W_M \cup W_N) \\ \text{out}(\text{extend}(Q, W_N)) &= W_M \cup W_N \end{aligned}$$

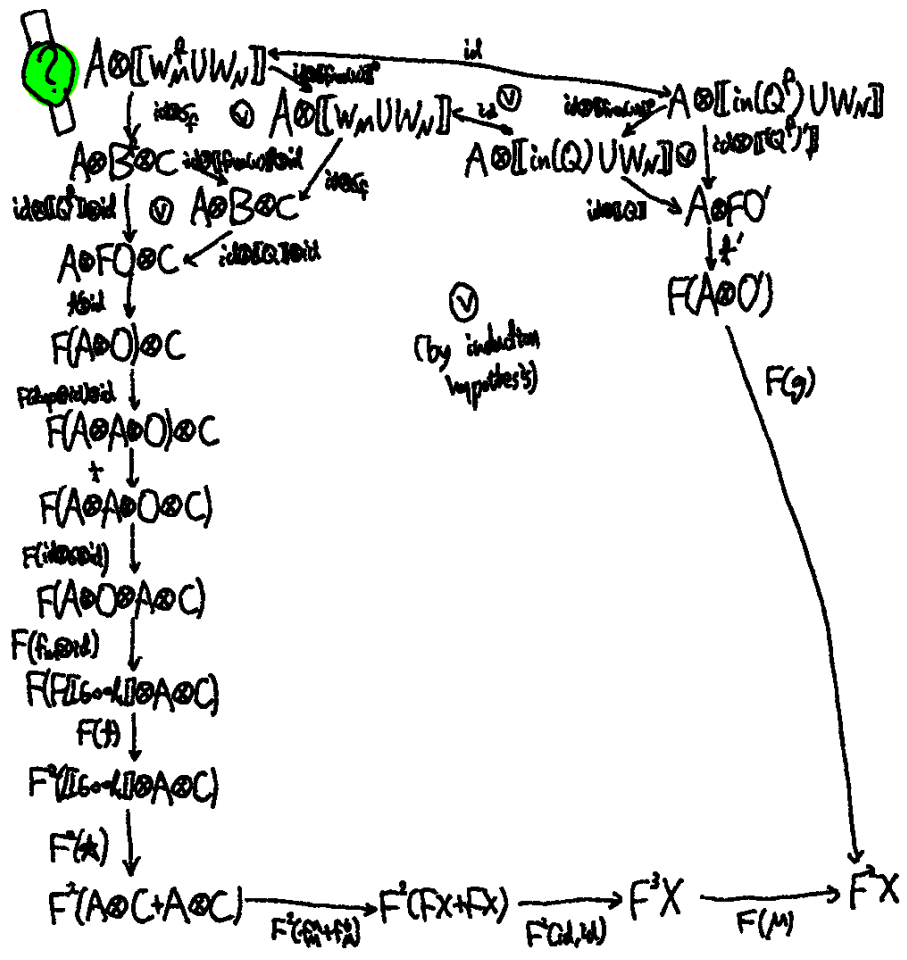
$$\begin{aligned} \mathcal{G} &= [\nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \text{if } m \text{ then } M_a \text{ else } M_b, A)] \\ &= [!\Delta] \circ [W_M \cup W_N] \\ &\quad \downarrow \text{id} \circ \sigma_f \\ &= [!\Delta] \circ [W_M] \circ [W_N] \\ &\quad \downarrow \text{dup} \circ \text{id} \\ &= [!\Delta] \circ [!\Delta] \circ [W_M] \circ [W_N] \\ &\quad \downarrow \text{id} \circ \sigma_{\text{id}} \\ &= [!\Delta] \circ [W_M] \circ [!\Delta] \circ [W_N] \\ &\quad \downarrow f_m \circ \text{id} \\ &= F(\text{bool}) \circ [!\Delta] \circ [W_N] \\ &\quad \downarrow \ddagger \\ &= F(\text{bool}) \circ [!\Delta] \circ [W_N] \\ &\quad \downarrow F(\star) \\ &= F([!\Delta] \circ [W_N] + [!\Delta] \circ [W_N]) \\ &\quad \downarrow F(f_m^a, f_m^b) \\ &= F^2[!A] \\ &\quad \downarrow M \\ &= F!A \end{aligned}$$



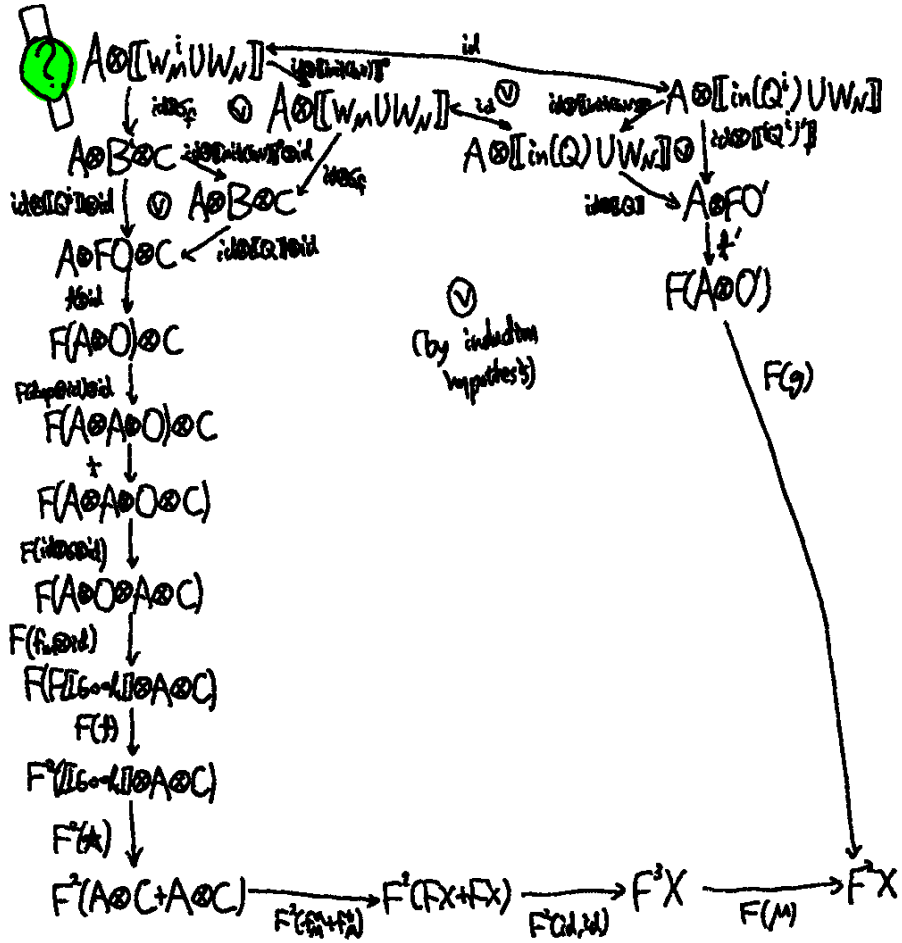
$$\begin{aligned}
 \textcircled{2} \quad Q^U &= U(V) Q, \quad Q' = \text{extend}(Q, W_N) \\
 (Q^U)' &= \text{extend}(Q^U, W_N) = U(V) Q' \\
 \text{in}(Q^U) &= W_M^U \\
 \text{out}(Q^U) &= \text{out}(Q), \quad \text{out}(\text{extend}(Q^U, W_N)) = \text{out}(\text{extend}(Q, W_N))
 \end{aligned}$$



③ $Q^f = \text{free } v \ Q, \quad Q' = \text{extend}(Q, W_N)$
 $(Q^f)' = \text{extend}(Q^f, W_N) = \text{free } v \ Q'$
 $\text{in}(Q^f) = W_M^f$
 $\text{out}(Q^f) = \text{out}(Q), \text{ out}((Q^f)') = \text{out}(Q')$



④ $Q^i = \text{init } b \vee Q, Q' = \text{extend}(Q, W_N)$
 $(Q^i)' = \text{extend}(Q^i, W_N) = \text{init } b \vee Q'$
 $\text{in}(Q^i) = W_N^i$
 $\text{out}(Q^i) = \text{out}(Q), \text{out}((Q^i)') = \text{out}(Q')$



$$\textcircled{5} Q^m = \text{meas} \vee Q_1, Q_2,$$

$$Q_1' = \text{extend}(Q_1, W_M), \quad Q_2' = \text{extend}(Q_2, W_M)$$

$$Q^m' = \text{extend}(Q^m, W_M) \\ = \text{meas} \vee Q_1', Q_2'$$

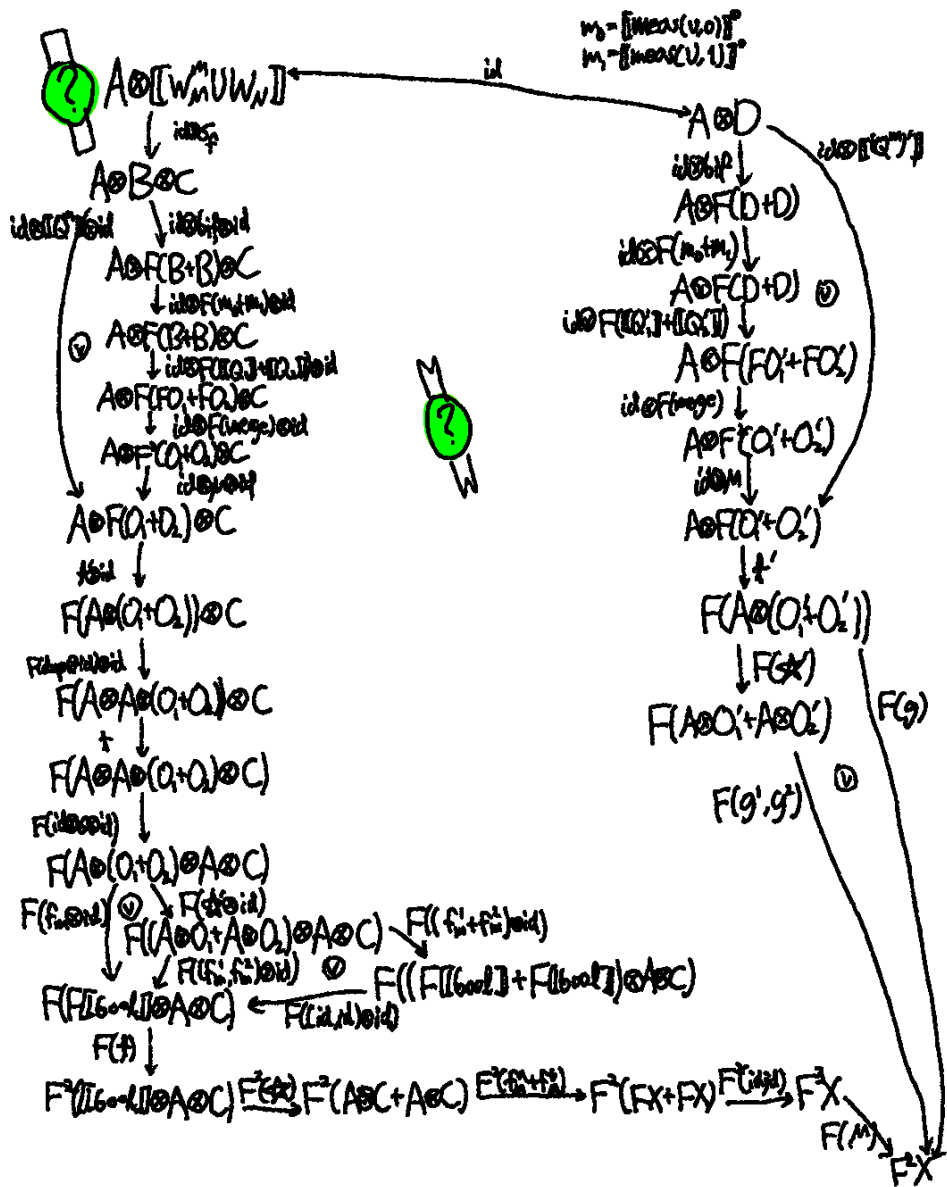
$$\text{out}(Q^m) = [\text{out}(Q_1), \text{out}(Q_2)] \\ \text{out}(Q^m') = [\text{out}(Q_1'), \text{out}(Q_2')] \\ \text{in}(Q^m) = W_M^m = \text{in}(Q_1) = \text{in}(Q_2)$$

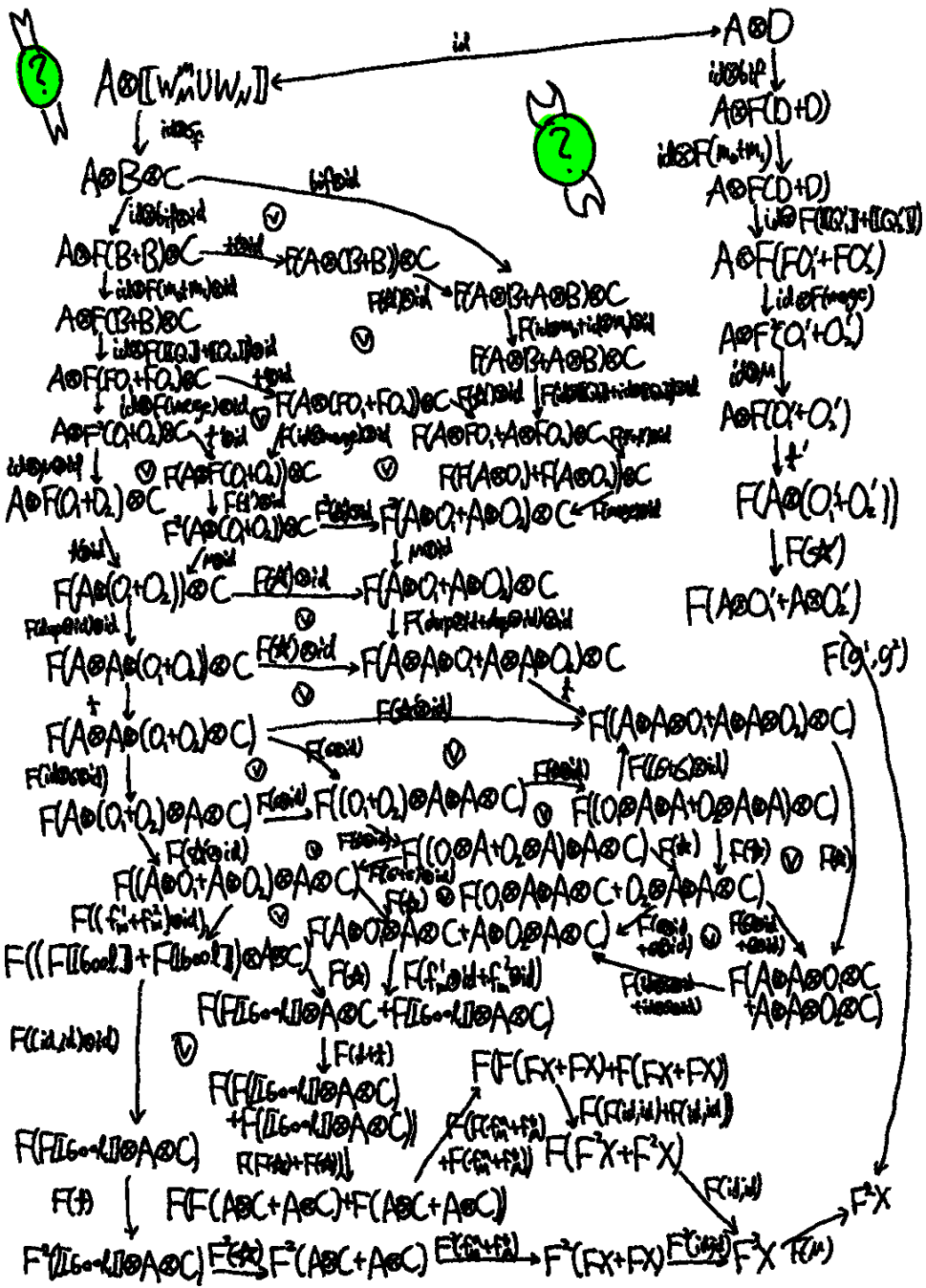
$$\begin{aligned} \llbracket Q^m \rrbracket &= \llbracket \text{in}(\text{meas} \vee Q_1, Q_2) \rrbracket \\ &\downarrow \text{bit} \\ &F(\llbracket \text{in}(\text{meas} \vee Q_1, Q_2) \rrbracket \\ &\quad + \llbracket \text{in}(\text{meas} \vee Q_1, Q_2) \rrbracket) \\ &\downarrow F(\llbracket Q_1 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket^* \\ &\quad + \llbracket Q_2 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket^*) \\ &F(F\llbracket \text{out}(Q_1) \rrbracket + F\llbracket \text{out}(Q_2) \rrbracket) \\ &\downarrow F(\text{merge}) \\ &F^2(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \\ &\downarrow M \\ &F(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \end{aligned}$$

$$\begin{aligned} f_m &= \llbracket !\Delta \rrbracket \circ (\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \\ &\downarrow \star' \\ &\llbracket !\Delta \rrbracket \circ \llbracket \text{out}(Q_1) \rrbracket + \llbracket !\Delta \rrbracket \circ \llbracket \text{out}(Q_2) \rrbracket \\ &\downarrow (f_m^1, f_m^2) \\ &F\llbracket \text{bool} \rrbracket \end{aligned}$$

$$\begin{aligned} g &= \llbracket !\Delta \rrbracket \circ (\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \\ &\downarrow \star' \\ &\llbracket !\Delta \rrbracket \circ \llbracket \text{out}(Q_1) \rrbracket + \llbracket !\Delta \rrbracket \circ \llbracket \text{out}(Q_2) \rrbracket \\ &\downarrow (g^1, g^2) \\ &F\llbracket A \rrbracket \end{aligned}$$

$$\begin{aligned} \llbracket Q^m' \rrbracket &= \llbracket \text{in}(Q^m') \rrbracket \\ &\downarrow \text{bit} \\ &F(\llbracket \text{in}(Q^m') \rrbracket \\ &\quad + \llbracket \text{in}(Q^m') \rrbracket) \\ &\downarrow F(\llbracket Q_1 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket^* \\ &\quad + \llbracket Q_2 \rrbracket \circ \llbracket \text{meas}(u, v) \rrbracket^*) \\ &F(F\llbracket \text{out}(Q_1) \rrbracket + F\llbracket \text{out}(Q_2) \rrbracket) \\ &\downarrow F(\text{merge}) \\ &F^2(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \\ &\downarrow M \\ &F(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket) \end{aligned}$$





A.12 . Congruence: let-construct

$$\frac{(\varepsilon(W_M), M) \rightarrow (Q, m) \quad \text{all}(Q) \cap W_N = \emptyset}{(\varepsilon(W_M \cup W_N), \text{let } \langle x, y \rangle = M \text{ in } N) \rightarrow (\text{extend}(Q, W_N), \text{let } \langle x, y \rangle = m \text{ in } N)}$$

$$\begin{aligned} & \llbracket !\Delta \vdash (\varepsilon(W_M \cup W_N), \text{let } \langle x, y \rangle = M \text{ in } N) : A \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket W_M \cup W_N \rrbracket \xrightarrow{\eta} F(\llbracket !\Delta \rrbracket \otimes \llbracket W_M \cup W_N \rrbracket) \xrightarrow{FG} F^2[A] \xrightarrow{M} F[A] \\ & f = \llbracket \nu \text{Bind}(!\Delta, W_M \cup W_N, \text{let } \langle x, y \rangle = M \text{ in } N, A) \rrbracket \end{aligned}$$

$$\begin{aligned} & \llbracket !\Delta \vdash (\text{extend}(Q, W_N), \text{let } \langle x, y \rangle = m \text{ in } N) : A \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \cup W_N \rrbracket \xrightarrow{\text{id} \otimes \llbracket \text{out}(Q, W_N) \rrbracket} \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q, W_N) \rrbracket) \end{aligned}$$

$$\text{in}(\text{extend}(Q, W_N)) = \text{in}(Q) \cup W_N$$

$$\begin{aligned} & \downarrow \eta' \\ & F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q, W_N) \rrbracket) \\ & \downarrow FG \\ & F^2[A] \\ & \downarrow M \\ & F[A] \end{aligned}$$

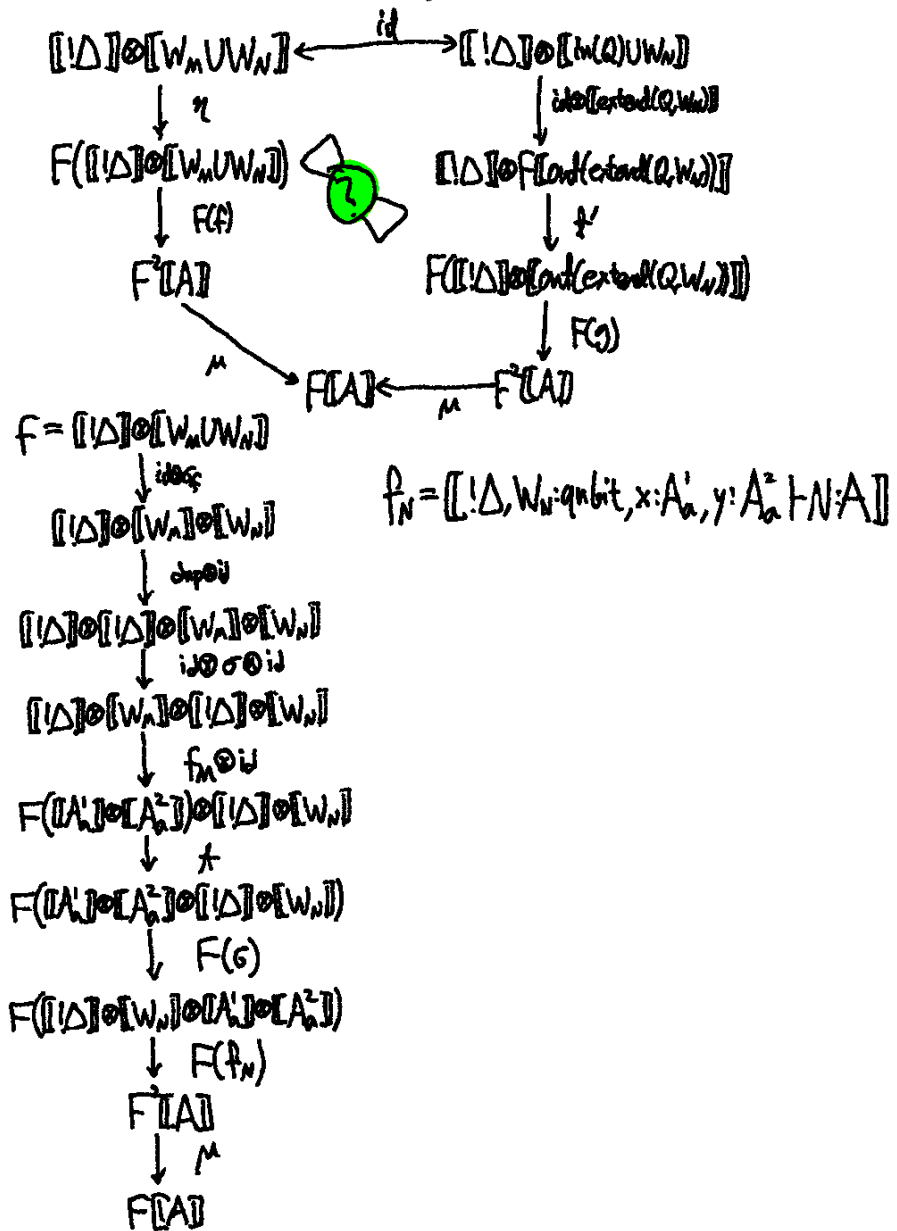
$$g = \llbracket \nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \text{let } \langle x, y \rangle = m \text{ in } N, A) \rrbracket$$

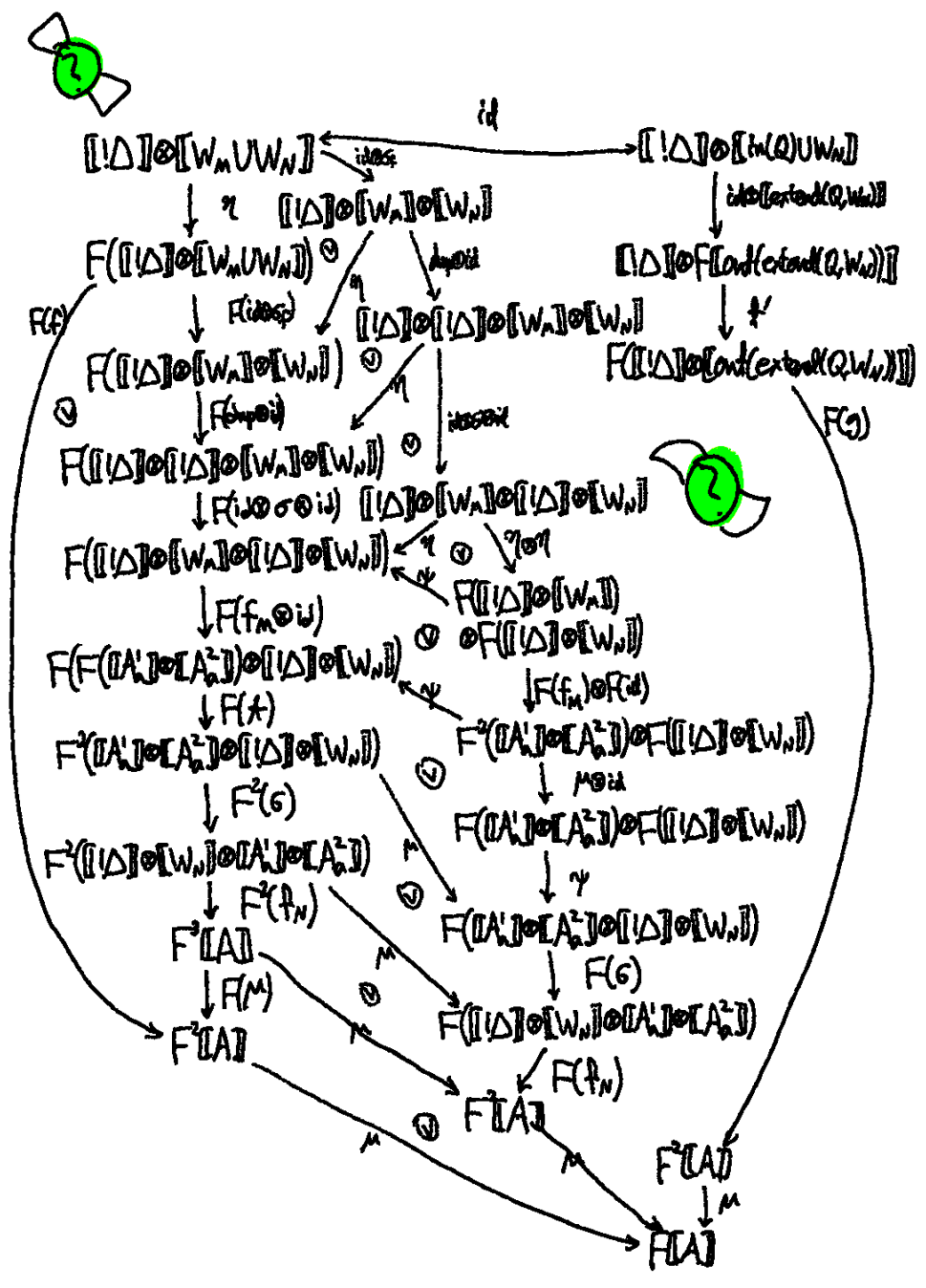
From the induction hypothesis:

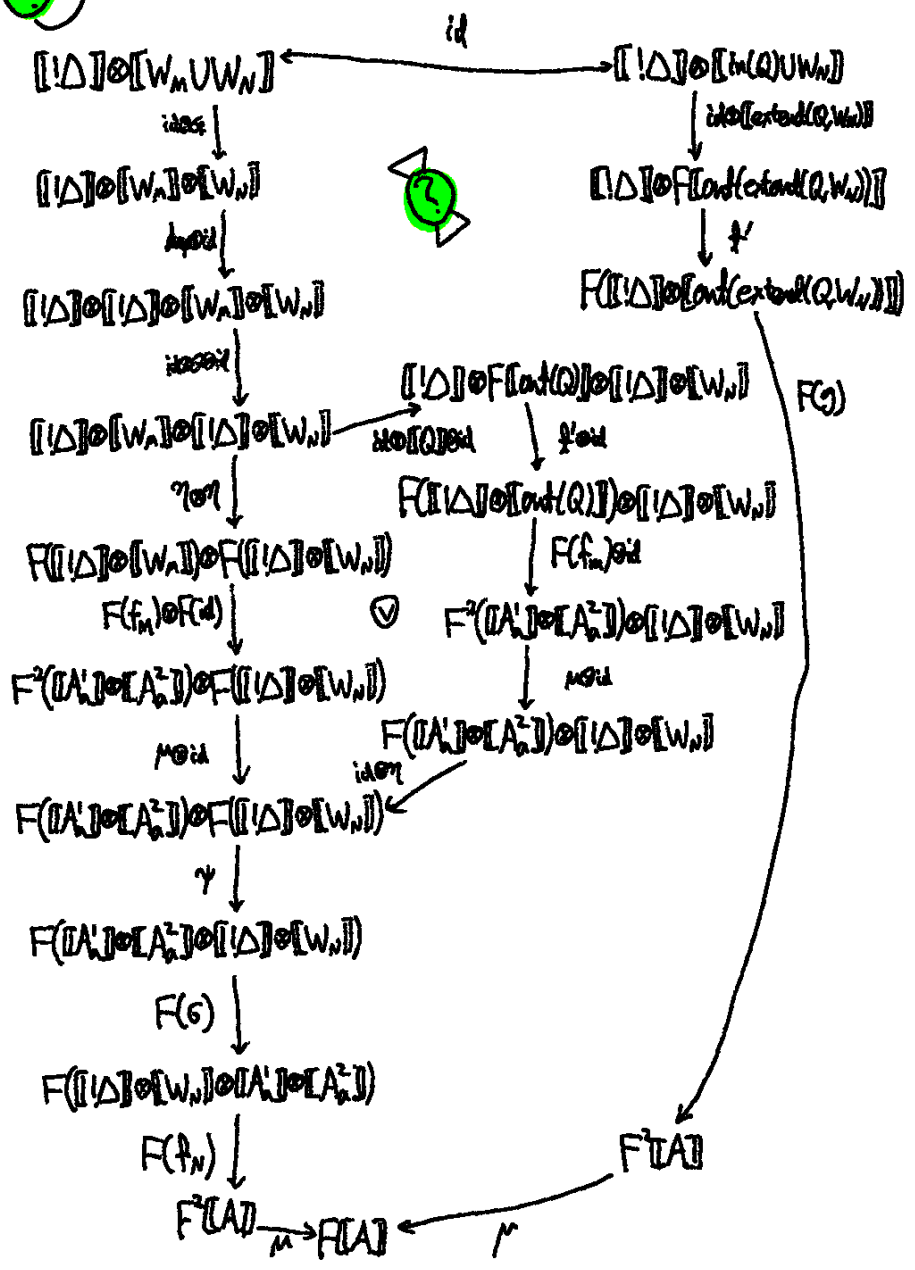
$$\begin{array}{ccc} \llbracket !\Delta \rrbracket \otimes \llbracket W_M \rrbracket & \longleftarrow & \llbracket !\Delta \rrbracket \otimes \llbracket \text{in}(Q) \rrbracket \\ \downarrow \eta & & \downarrow \text{id} \otimes \llbracket Q \rrbracket \\ F(\llbracket !\Delta \rrbracket \otimes \llbracket W_M \rrbracket) & & \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q) \rrbracket) \\ \downarrow FG_M & & \downarrow \eta' \\ F[A_M] & & F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q) \rrbracket) \\ \downarrow M & & \downarrow FG_m \\ F[A] & \longleftarrow & F^2[A] \end{array}$$

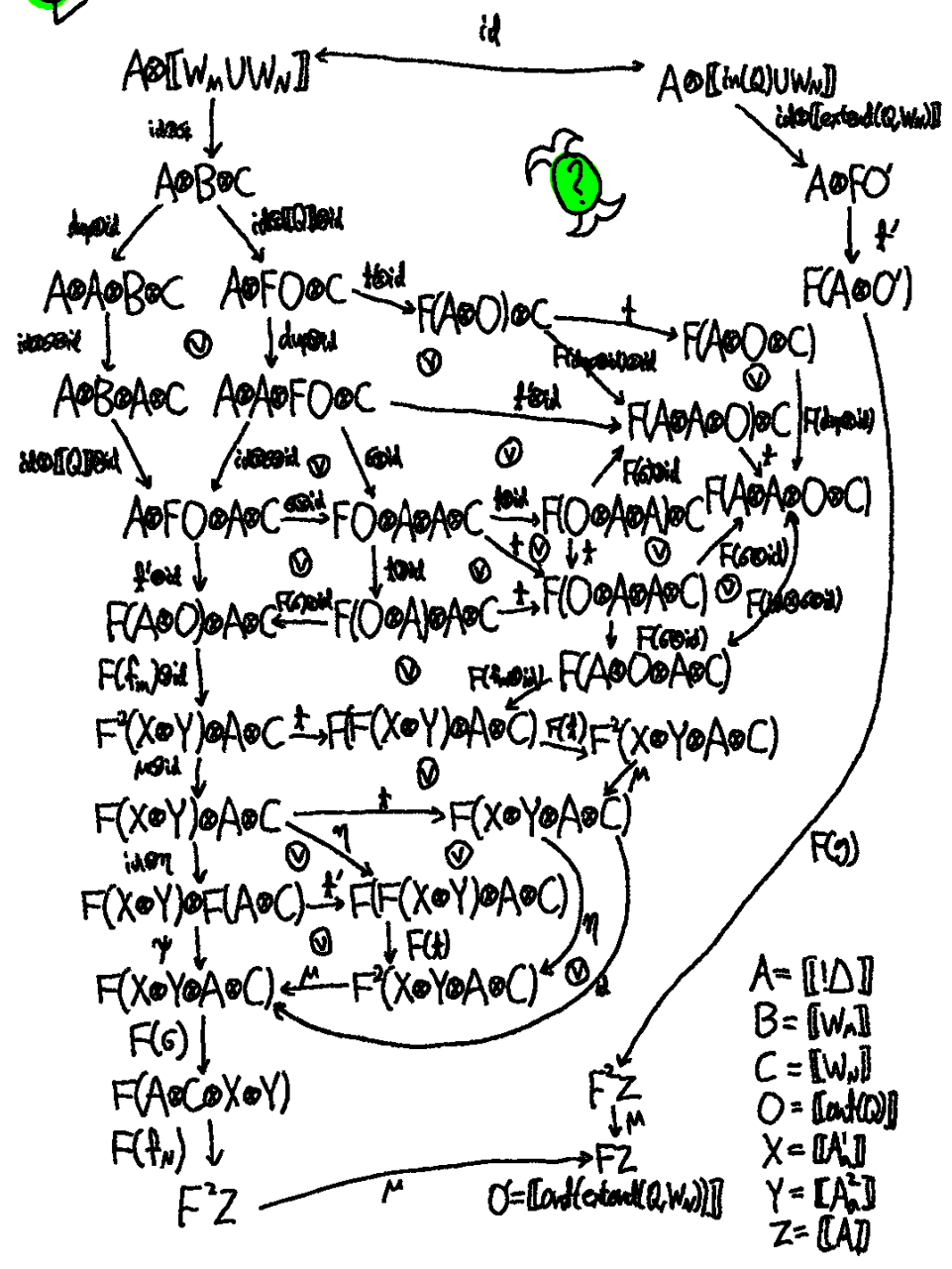
$f_M = \llbracket \nu \text{Bind}(!\Delta, W_M, M, A_M) \rrbracket$
 $f_m = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q), m, A_m) \rrbracket$
 $A_m = A_M \otimes A_Q^2$

We show the following:



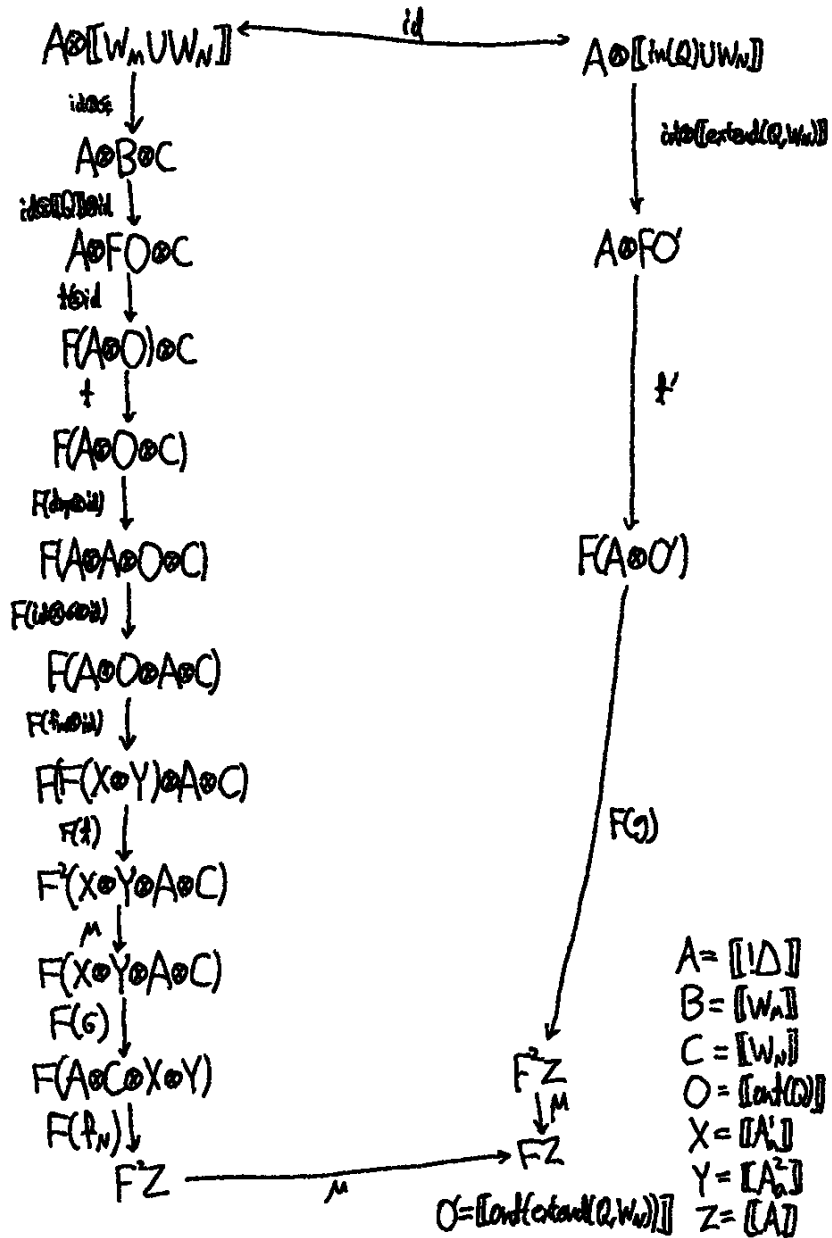








We prove the following by induction on Q .

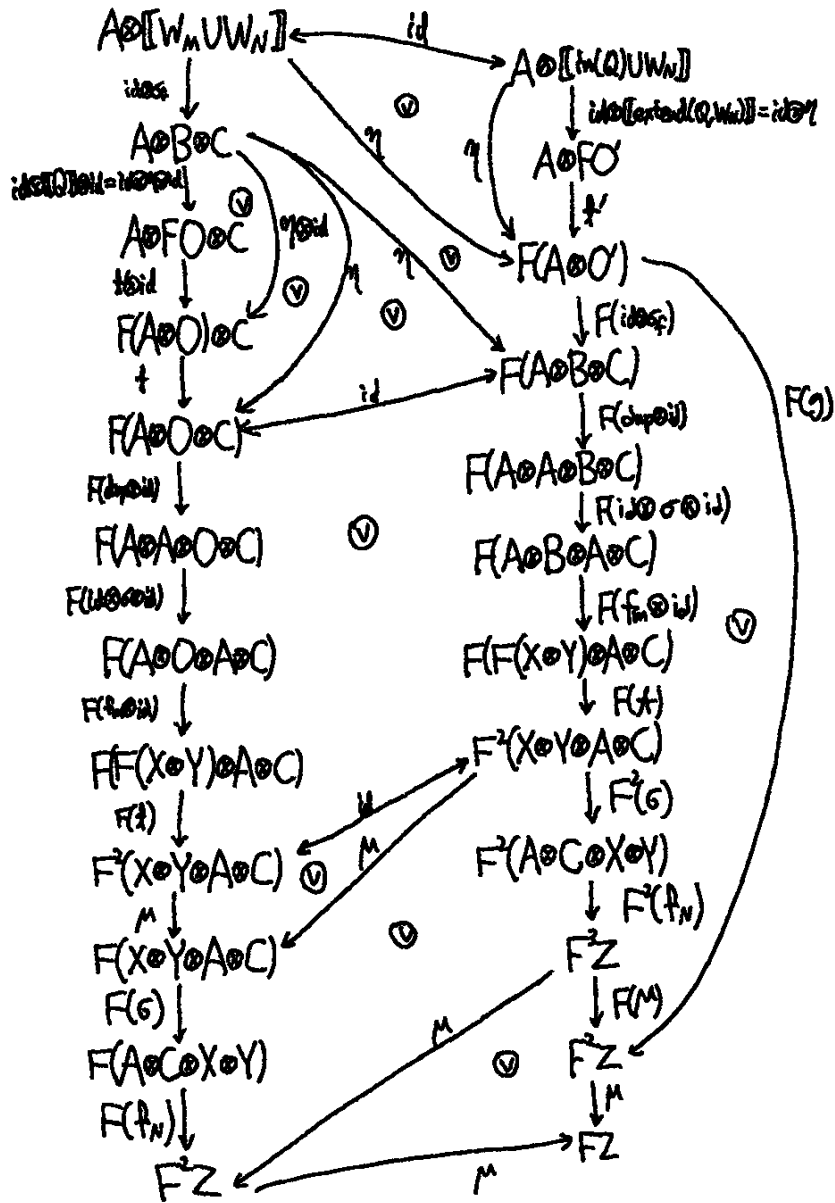


$$\textcircled{1} \quad Q = \Sigma(W_N) \quad \text{extend}(Q, W_N) = \Sigma(W_N \cup W_N)$$

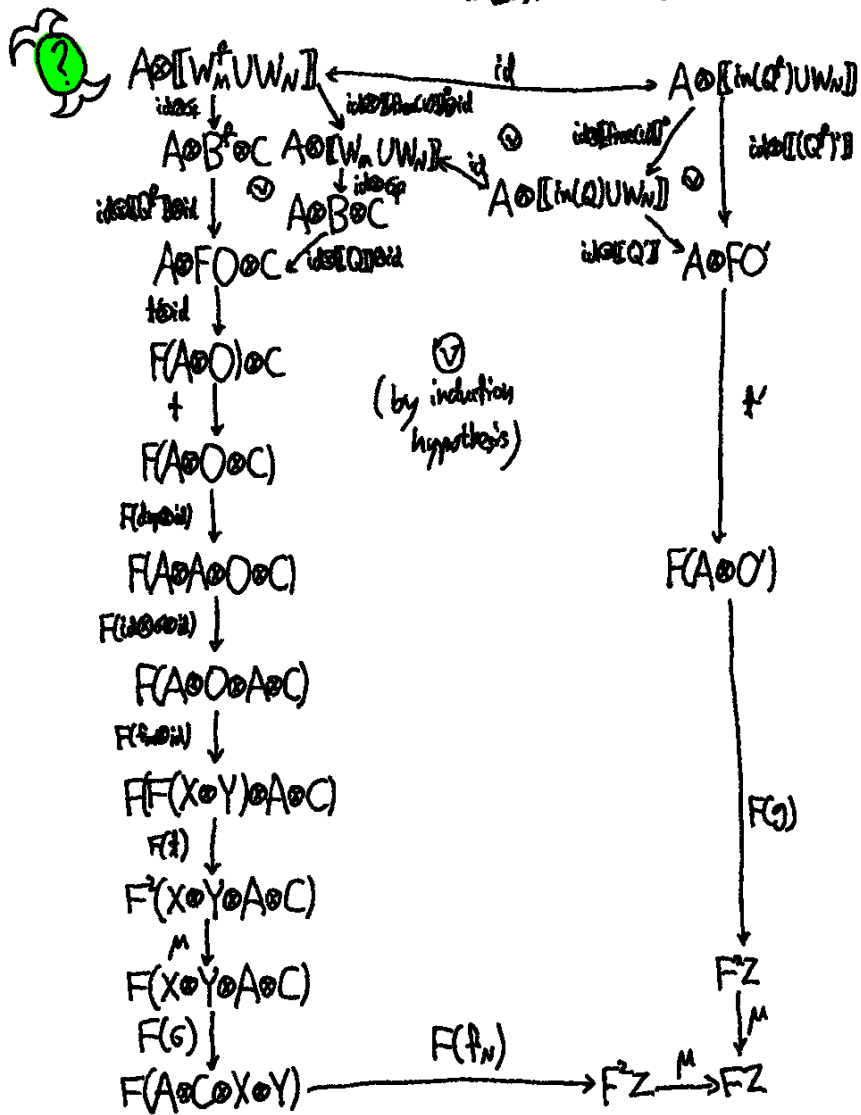
$$\text{in}(Q) = W_N$$

$$\text{out}(Q) = W_N$$

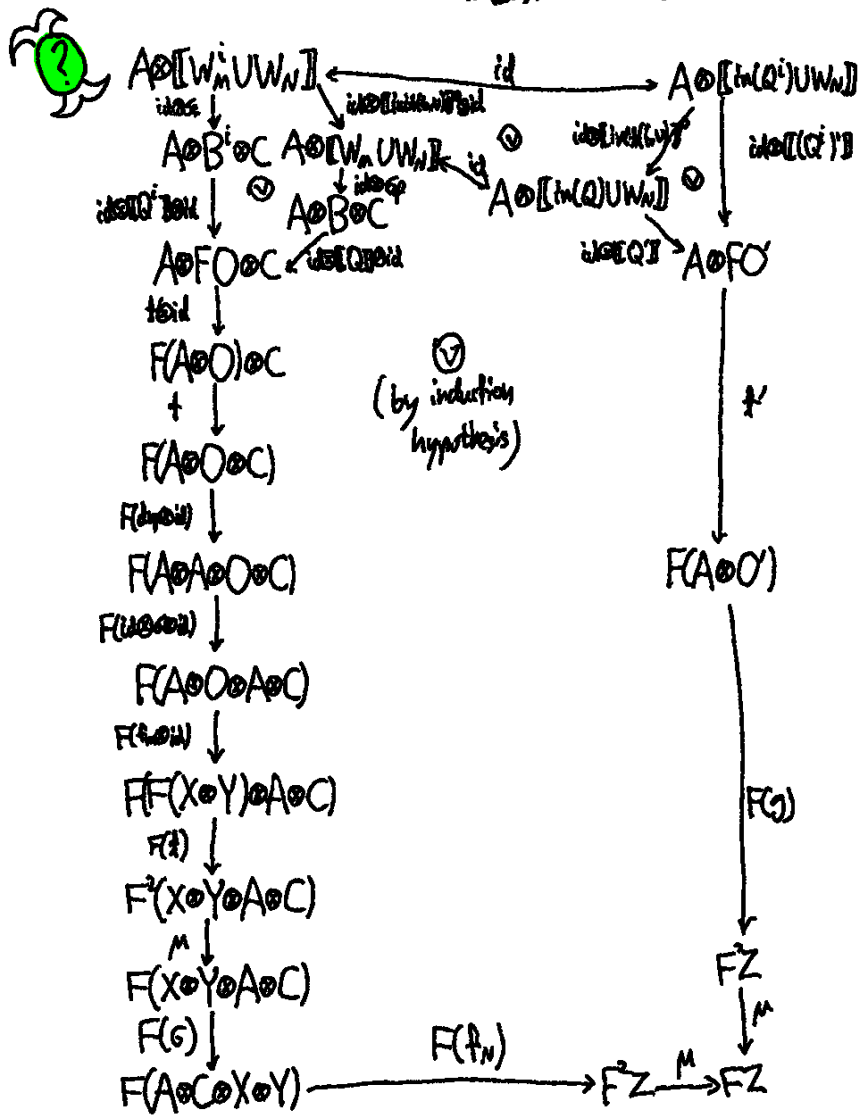
$$\begin{aligned}
 g &= [\nu \text{Bind}(!\Delta, \text{out}(\text{extend}(Q, W_N)), \text{let } \langle x, y \rangle = m \text{ in } N, A)] \\
 &= [!\Delta] \circ [W_N \cup W_N] \\
 &\quad \downarrow \text{id} \circ \sigma \\
 &= [!\Delta] \circ [W_N] \circ [W_N] \\
 &\quad \downarrow \text{dup} \\
 &= [!\Delta] \circ [!\Delta] \circ [W_N] \circ [W_N] \\
 &\quad \downarrow \text{id} \circ \sigma \circ \text{id} \\
 &= [!\Delta] \circ [W_N] \circ [!\Delta] \circ [W_N] \\
 &\quad \downarrow \text{f} \circ \text{id} \\
 &= F([!\Delta] \circ [A_1^2]) \circ [!\Delta] \circ [W_N] \\
 &\quad \downarrow * \\
 &= F([!\Delta] \circ [A_1^2]) \circ [!\Delta] \circ [W_N] \\
 &\quad \downarrow F(\sigma) \\
 &= F([!\Delta] \circ [W_N] \circ [!\Delta] \circ [A_1^2]) \\
 &\quad \downarrow F(\tau_N) \\
 &= F[!\Delta] \\
 &\quad \downarrow M \\
 &= F[A]
 \end{aligned}$$



$$\begin{aligned} \textcircled{3} \quad Q^f &= \text{free } v \ Q \quad \llbracket Q \rrbracket = \llbracket Q \rrbracket \cdot \llbracket \text{free } v \rrbracket \\ (Q^f)' &= \text{extend}(Q^f, W_N) = \text{free } v \ Q' \quad \llbracket (Q^f)' \rrbracket = \llbracket Q' \rrbracket \cdot \llbracket \text{free } v \rrbracket \\ Q' &= \text{extend}(Q, W_N) \quad \begin{aligned} \text{in}(Q^f) &= W_N^f, \text{out}(Q^f) = \text{out}(Q) \\ \text{in}(Q') &= \text{in}(Q) \cup W_N, \text{out}(Q') = \text{out}(Q) \end{aligned} \end{aligned}$$



$$\begin{aligned}
 \textcircled{4} \quad Q^i &= \text{init } b \vee Q & \llbracket Q^i \rrbracket &= \llbracket Q \rrbracket \circ \llbracket \text{init}(b, v) \rrbracket^* \\
 (Q^i)' &= \text{extend}(Q^i, W_N) = \text{init } b \vee Q' & \llbracket (Q^i)' \rrbracket &= \llbracket Q^i \rrbracket \circ \llbracket \text{init}(b, v) \rrbracket^* \\
 Q' &= \text{extend}(Q, W_N) & \text{in}(Q^i) &= W_N^i, \text{out}(Q^i) = \text{out}(Q) \\
 & & \text{in}((Q^i)') &= \text{in}(Q^i) \cup W_N, \text{out}((Q^i)') = \text{out}(Q')
 \end{aligned}$$



$$\textcircled{5} \quad Q^m = \text{meas} \vee Q_1, Q_2$$

$$(Q^m)' = \text{extend}(Q^m, W_N)$$

$$= \text{meas} \vee Q_1', Q_2'$$

$$\text{in}(Q^m) = W_N, \text{out}(Q^m) = [\text{out}(Q_1), \text{out}(Q_2)]$$

$$\text{in}(Q^m)' = \text{in}(Q^m) \vee W_N, \text{out}(Q^m)' = [\text{out}(Q_1'), \text{out}(Q_2)']$$

$$Q_1' = \text{extend}(Q_1, W_N)$$

$$Q_2' = \text{extend}(Q_2, W_N)$$

$$[[Q^m]] = [[\text{in}(\text{meas} \vee Q_1, Q_2)]]$$

$$\downarrow \text{bit}$$

$$F([\text{in}(\text{meas} \vee Q_1, Q_2)] + [\text{in}(\text{meas} \vee Q_1, Q_2)])$$

$$\downarrow F([\text{Q}_1] \circ [\text{meas}(u, 0)]^n + [\text{Q}_2] \circ [\text{meas}(u, 1)]^n)$$

$$F(F[\text{out}(Q_1)] + F[\text{out}(Q_2)])$$

$$\downarrow \text{FC merge}$$

$$F^2([\text{out}(Q_1)] + [\text{out}(Q_2)])$$

$$\downarrow M$$

$$F([\text{out}(Q_1)] + [\text{out}(Q_2)])$$

$$f_m = [[!\Delta]] \circ ([\text{out}(Q_1)] + [\text{out}(Q_2)])$$

$$\downarrow \star'$$

$$[[!\Delta]] \circ [\text{out}(Q_1)] + [[!\Delta]] \circ [\text{out}(Q_2)]$$

$$\downarrow (f_m^1, f_m^2)$$

$$F([\Lambda_n^1] \circ [\Lambda_n^2])$$

$$g = [[!\Delta]] \circ ([\text{out}(Q_1')] + [\text{out}(Q_2)'])$$

$$\downarrow \star'$$

$$[[!\Delta]] \circ [\text{out}(Q_1')] + [[!\Delta]] \circ [\text{out}(Q_2)']$$

$$\downarrow (g^1, g^2)$$

$$F[\Lambda]$$

$$[[Q^m]'] = [[\text{in}(Q^m)']]$$

$$\downarrow \text{bit}$$

$$F([\text{in}(Q^m)'] + [\text{in}(Q^m)'])$$

$$\downarrow F([\text{Q}_1] \circ [\text{meas}(u, 0)]^n + [\text{Q}_2] \circ [\text{meas}(u, 1)]^n)$$

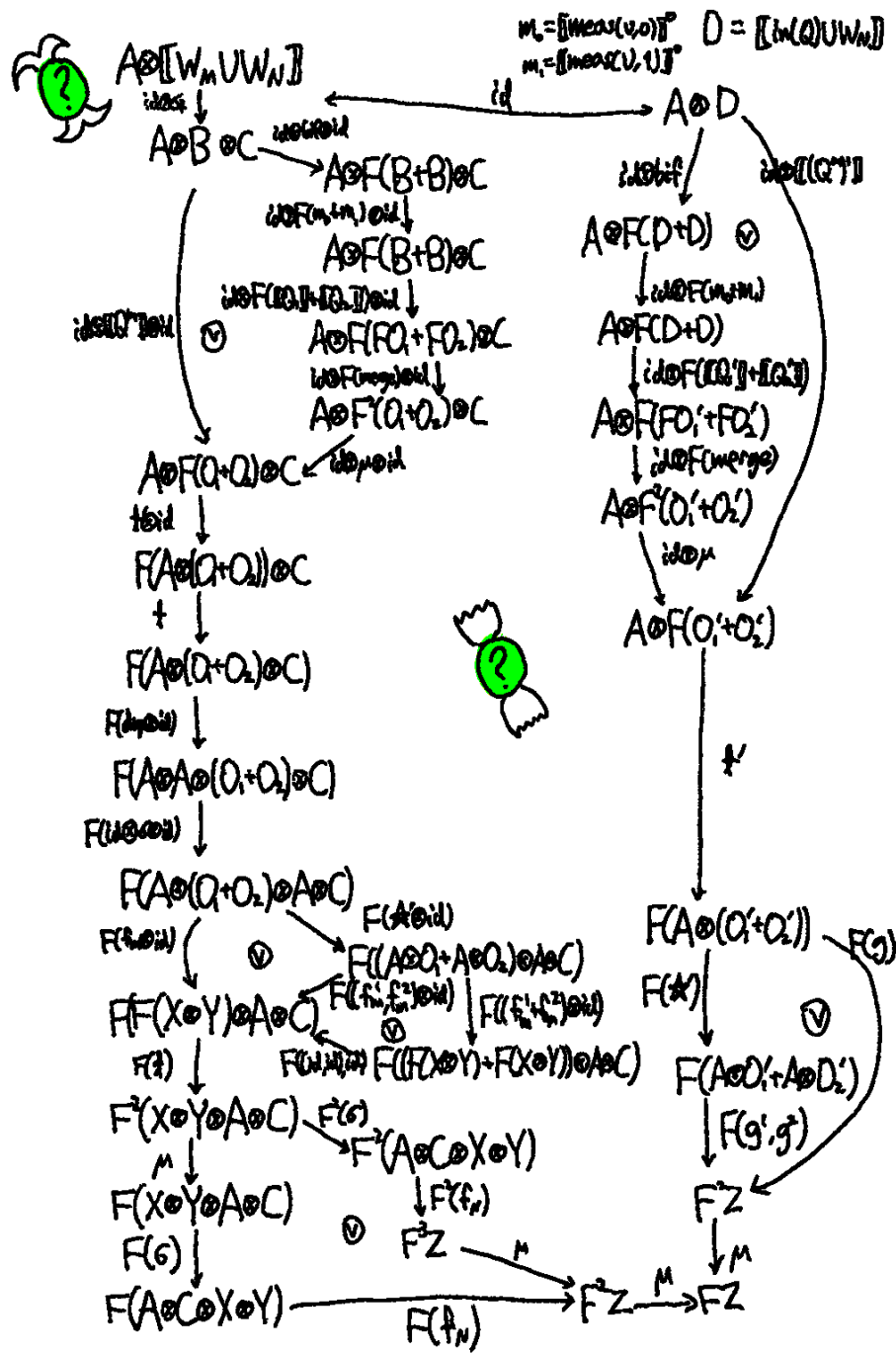
$$F(F[\text{out}(Q_1')] + F[\text{out}(Q_2)'])$$

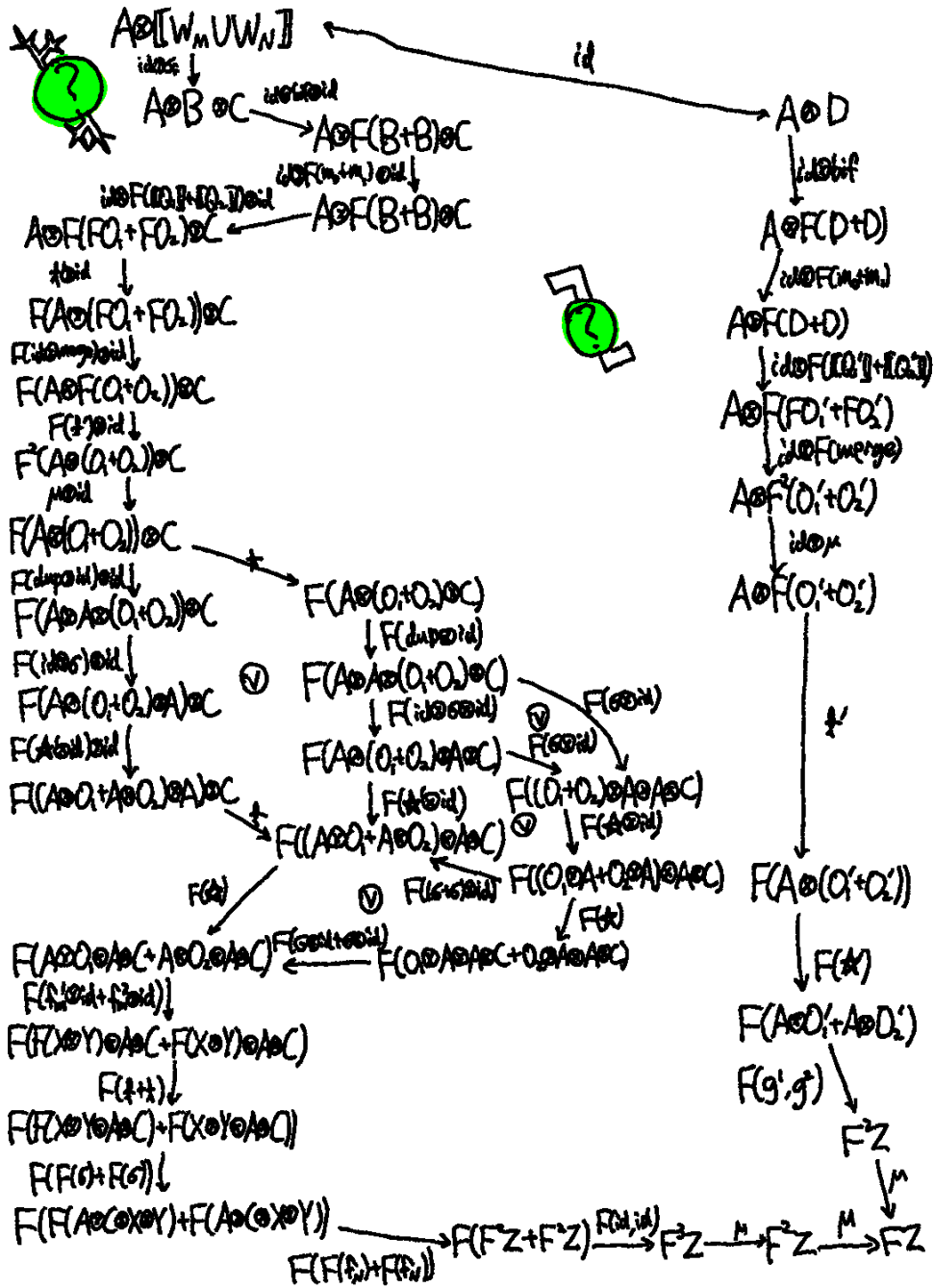
$$\downarrow \text{FC merge}$$

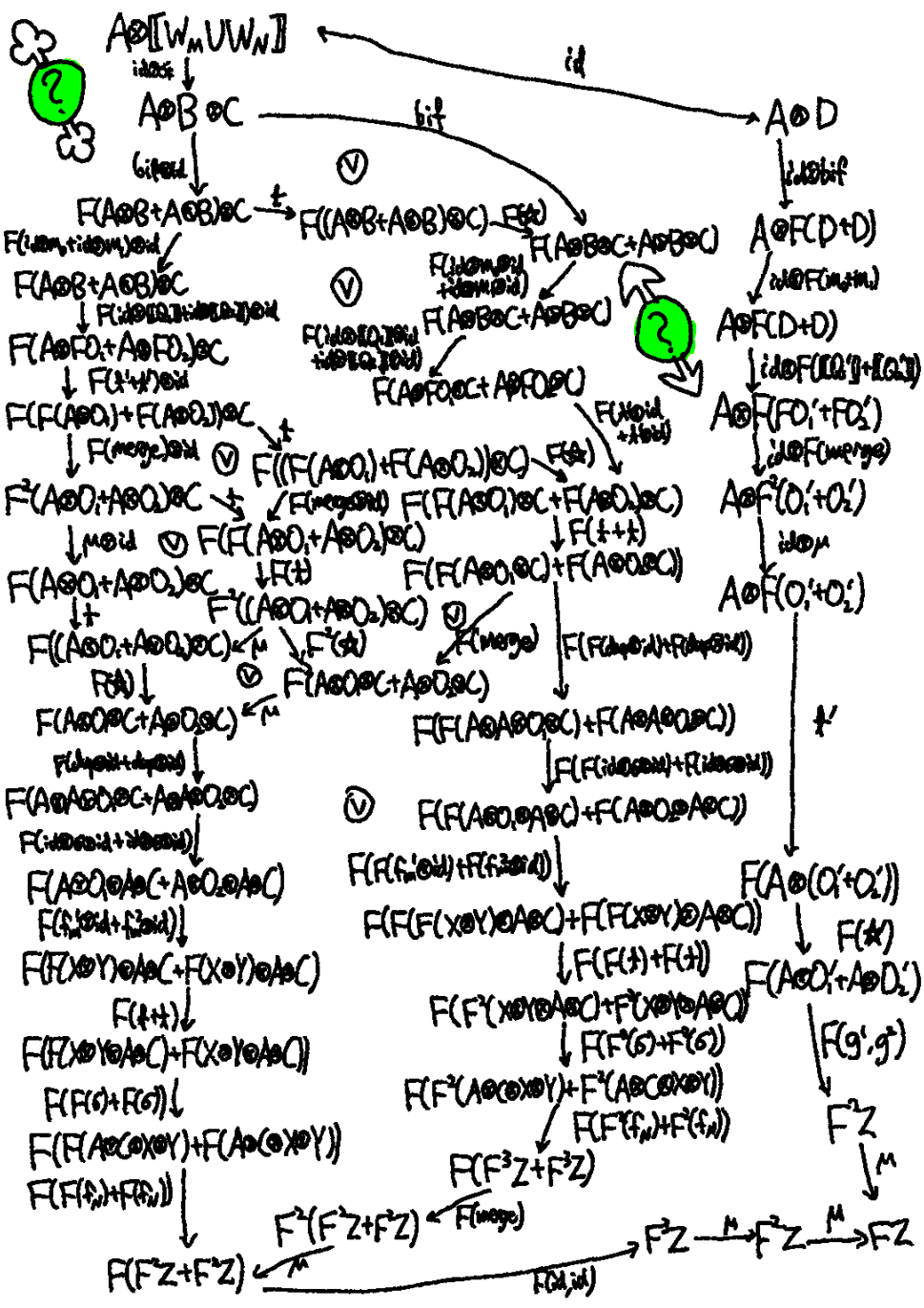
$$F^2([\text{out}(Q_1')] + [\text{out}(Q_2)'])$$

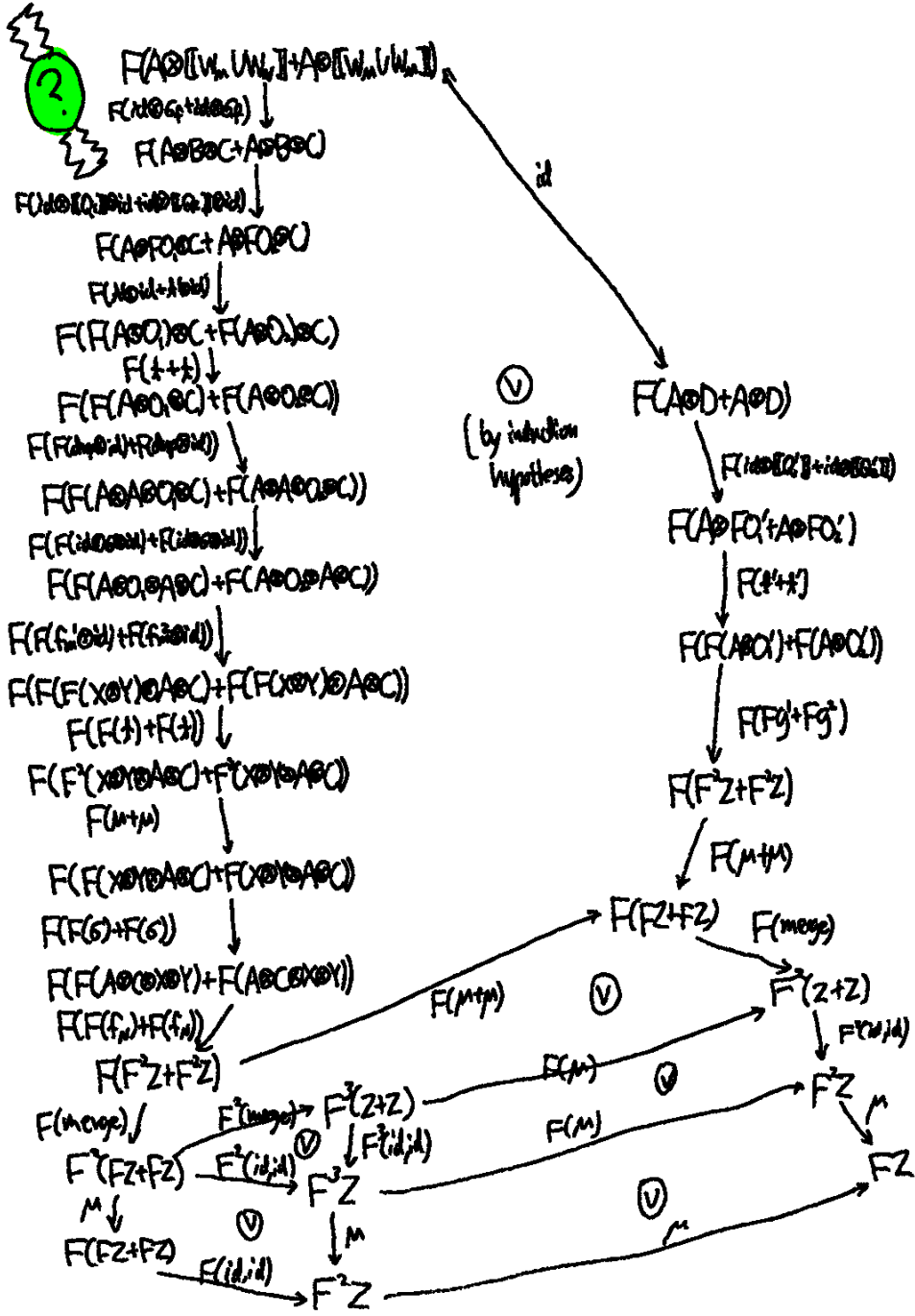
$$\downarrow M$$

$$F([\text{out}(Q_1')] + [\text{out}(Q_2)'])$$









A.13 . Congruence: branching, case 1

$$\frac{(Q_1, m_a) \rightarrow (Q_3, m_c) \quad (Q_2, m_b) \rightarrow (Q_4, m_d)}{(\text{meas } w \ Q_1 \ Q_2), [m_a, m_b] \rightarrow (\text{meas } w \ Q_3 \ Q_4), [m_c, m_d]}$$

$$\llbracket !\Delta \vdash (\text{meas } w \ Q_1 \ Q_2), [m_a, m_b] : A \rrbracket$$

$$= \llbracket ([\text{meas } w \ Q_1 \ Q_2], f_{v\text{Bind}}) \rrbracket$$

$$= \llbracket !\Delta \rrbracket \otimes \llbracket W \rrbracket \xrightarrow{(\llbracket !\Delta \rrbracket) \otimes (\llbracket W \rrbracket)} \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket)$$

$$\begin{aligned} W^1 &= \text{in}(\text{meas } w \ Q_1 \ Q_2) \\ &= \text{in}(Q_1) = \text{in}(Q_2) \\ Q^1 &= \text{meas } w \ Q_1 \ Q_2 \end{aligned}$$

$$\downarrow \quad \downarrow$$

$$F(\llbracket !\Delta \rrbracket \otimes (\llbracket \text{out}(Q_1) \rrbracket + \llbracket \text{out}(Q_2) \rrbracket))$$

$$\downarrow_{\text{M}} F(f_{v\text{Bind}})$$

$$F^2 \llbracket A \rrbracket$$

$$\downarrow_{\text{M}}^M$$

$$F \llbracket A \rrbracket$$

$$f_{v\text{Bind}}^1 = \llbracket v\text{Bind}(!\Delta, [\text{out}(Q_1), \text{out}(Q_2)], [m_a, m_b], A) \rrbracket$$

$$\llbracket !\Delta \vdash (\text{meas } w \ Q_3 \ Q_4), [m_c, m_d] : A \rrbracket$$

$$= \llbracket ([\text{meas } w \ Q_3 \ Q_4], g_{v\text{Bind}}) \rrbracket$$

$$= \llbracket !\Delta \rrbracket \otimes \llbracket W \rrbracket \xrightarrow{(\llbracket !\Delta \rrbracket) \otimes (\llbracket W \rrbracket)} \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q_3) \rrbracket + \llbracket \text{out}(Q_4) \rrbracket)$$

$$\begin{aligned} W^2 &= \text{in}(\text{meas } w \ Q_3 \ Q_4) \\ &= \text{in}(Q_3) = \text{in}(Q_4) \\ Q^2 &= \text{meas } w \ Q_3 \ Q_4 \end{aligned}$$

$$\downarrow \quad \downarrow$$

$$F(\llbracket !\Delta \rrbracket \otimes (\llbracket \text{out}(Q_3) \rrbracket + \llbracket \text{out}(Q_4) \rrbracket))$$

$$\downarrow_{\text{M}} F(f_{v\text{Bind}}^2)$$

$$F^2 \llbracket A \rrbracket$$

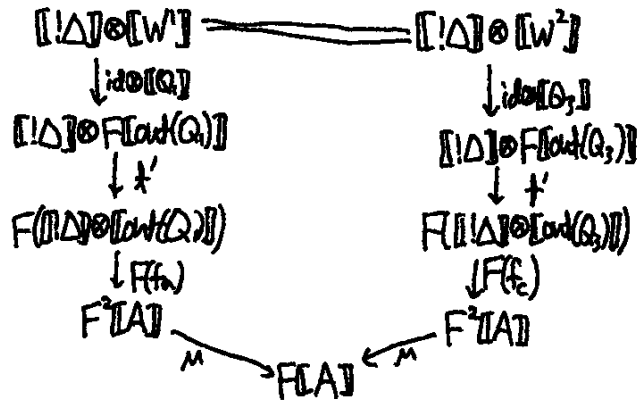
$$\downarrow_{\text{M}}^M$$

$$F \llbracket A \rrbracket$$

$$f_{v\text{Bind}}^2 = \llbracket v\text{Bind}(!\Delta, [\text{out}(Q_3), \text{out}(Q_4)], [m_c, m_d], A) \rrbracket$$

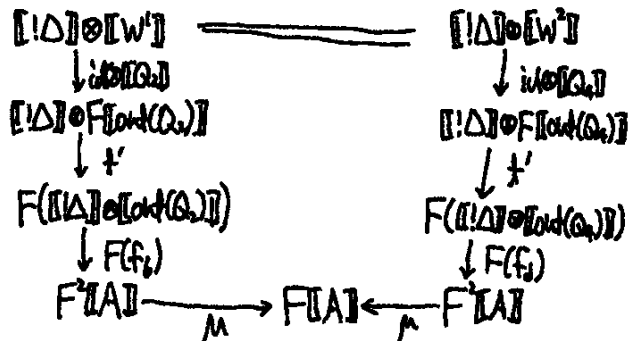
From the induction hypotheses:

$$[[! \Delta] \vdash (Q_1, m_a) : A] = [[! \Delta] \vdash (Q_3, m_c) : A]$$



$$f_a = [[\nu \text{Bind}(! \Delta, \text{out}(Q_1), m_a, A)]] \quad f_c = [[\nu \text{Bind}(! \Delta, \text{out}(Q_3), m_c, A)]]$$

$$[[! \Delta] \vdash (Q_2, m_b) : A] = [[! \Delta] \vdash (Q_4, m_d) : A]$$

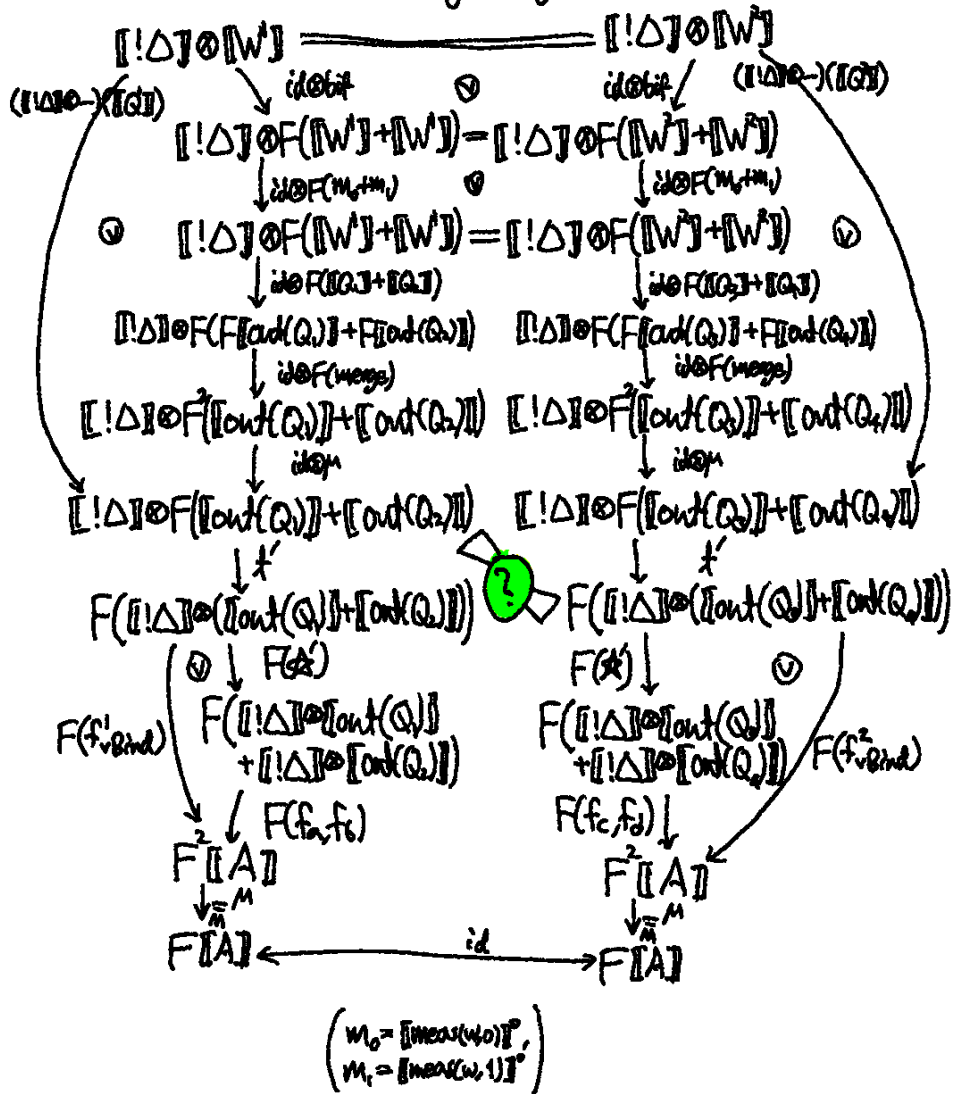


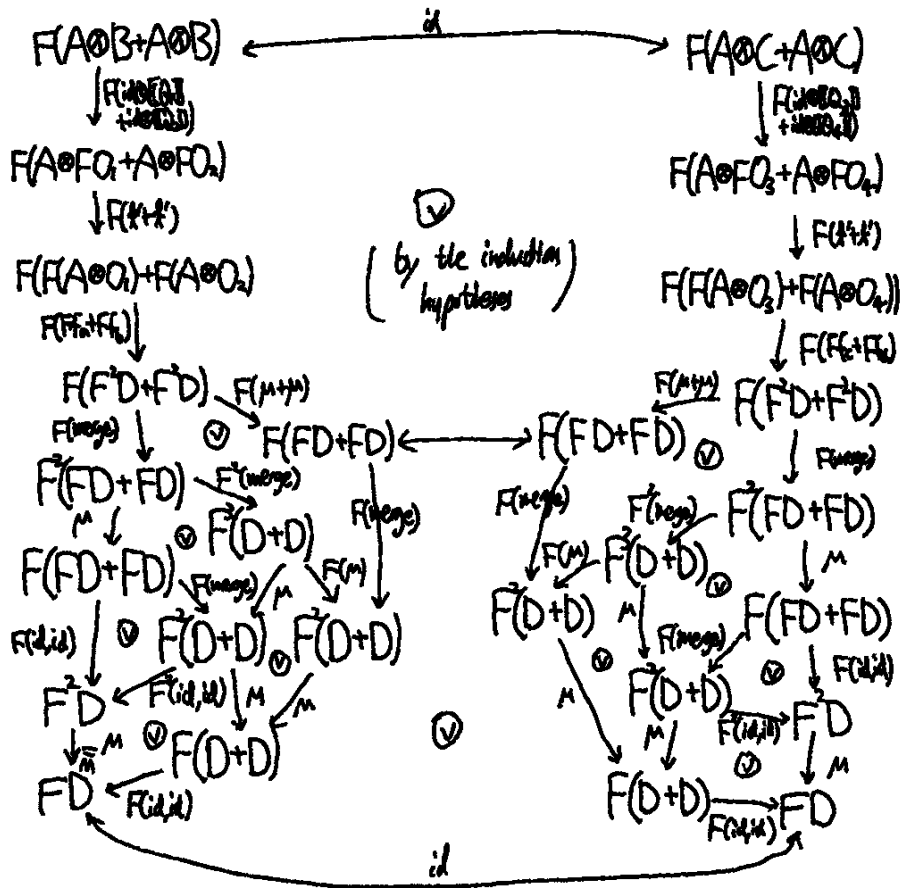
$$f_b = [[\nu \text{Bind}(! \Delta, \text{out}(Q_2), m_b, A)]] \quad f_d = [[\nu \text{Bind}(! \Delta, \text{out}(Q_4), m_d, A)]]$$

$$f_{\text{Bind}}^1 = [[! \Delta] \circ ([\text{out}(Q_1)] + [\text{out}(Q_2)])] \xrightarrow{\dagger} [[! \Delta] \circ [\text{out}(Q_1)] + [[! \Delta] \circ [\text{out}(Q_2)]] \xrightarrow{(f_a, f_c)} F[A]$$

$$f_{\text{Bind}}^2 = [[! \Delta] \circ ([\text{out}(Q_2)] + [\text{out}(Q_4)])] \xrightarrow{\dagger} [[! \Delta] \circ [\text{out}(Q_2)] + [[! \Delta] \circ [\text{out}(Q_4)]] \xrightarrow{(f_b, f_d)} F[A]$$

We show the following diagram commutes:





A.14 . Congruence: branching, case 2

$$\frac{(Q_1, m_a) \rightarrow (Q_3, m_c)}{((meas\ w\ Q_1\ Q_2), [m_a, v]) \rightarrow ((meas\ w\ Q_3\ Q_2), [m_c, v])}$$

$$\llbracket !\Delta \vdash ((meas\ w\ Q_1\ Q_2), [m_a, v]) : A \rrbracket$$

$$= \llbracket ((meas\ w\ Q_1\ Q_2), f_{vBind}^1) \rrbracket$$

$$= \llbracket !\Delta \rrbracket \otimes \llbracket W' \rrbracket \xrightarrow{(\llbracket \Delta \rrbracket) \otimes (\llbracket Q_2 \rrbracket)} \llbracket !\Delta \rrbracket \otimes F(\llbracket out(Q_1) \rrbracket + \llbracket out(Q_2) \rrbracket)$$

$$\downarrow \dagger$$

$$F(\llbracket !\Delta \rrbracket \otimes (\llbracket out(Q_1) \rrbracket + \llbracket out(Q_2) \rrbracket))$$

$$\downarrow_{\text{var}} F(f_{vBind}^1)$$

$$F^2 \llbracket A \rrbracket$$

$$\downarrow_{\text{var}}^M$$

$$F \llbracket A \rrbracket$$

$$Q_a = meas\ w\ Q_1\ Q_2$$

$$Q_b = meas\ w\ Q_3\ Q_2$$

$$W' = in(Q_a) = in(Q_1)$$

$$= in(Q_b) = in(Q_3)$$

$$f_{vBind}^1 = \llbracket vBind(!\Delta, [out(Q_1), out(Q_2)], [m_a, v], A) \rrbracket$$

$$\llbracket !\Delta \vdash ((meas\ w\ Q_3\ Q_2), [m_c, v]) : A \rrbracket$$

$$= \llbracket ((meas\ w\ Q_3\ Q_2), f_{vBind}^2) \rrbracket$$

$$= \llbracket !\Delta \rrbracket \otimes \llbracket W' \rrbracket \xrightarrow{(\llbracket \Delta \rrbracket) \otimes (\llbracket Q_2 \rrbracket)} \llbracket !\Delta \rrbracket \otimes F(\llbracket out(Q_3) \rrbracket + \llbracket out(Q_2) \rrbracket)$$

$$\downarrow \dagger$$

$$F(\llbracket !\Delta \rrbracket \otimes (\llbracket out(Q_3) \rrbracket + \llbracket out(Q_2) \rrbracket))$$

$$\downarrow_{\text{var}} F(f_{vBind}^2)$$

$$F^2 \llbracket A \rrbracket$$

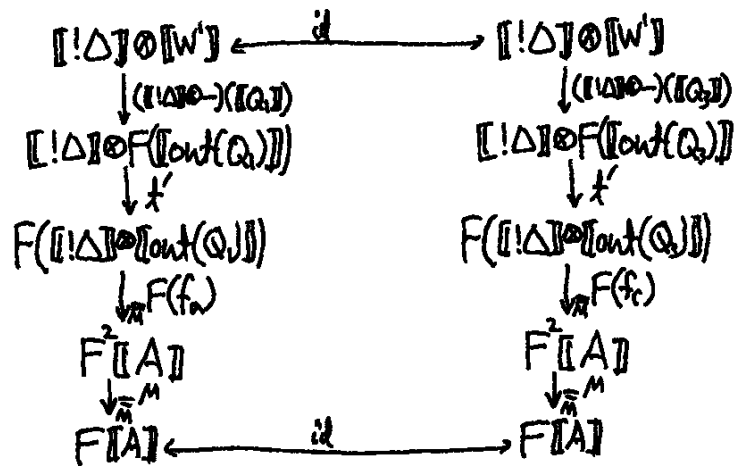
$$\downarrow_{\text{var}}^M$$

$$F \llbracket A \rrbracket$$

$$f_{vBind}^2 = \llbracket vBind(!\Delta, [out(Q_3), out(Q_2)], [m_c, v], A) \rrbracket$$

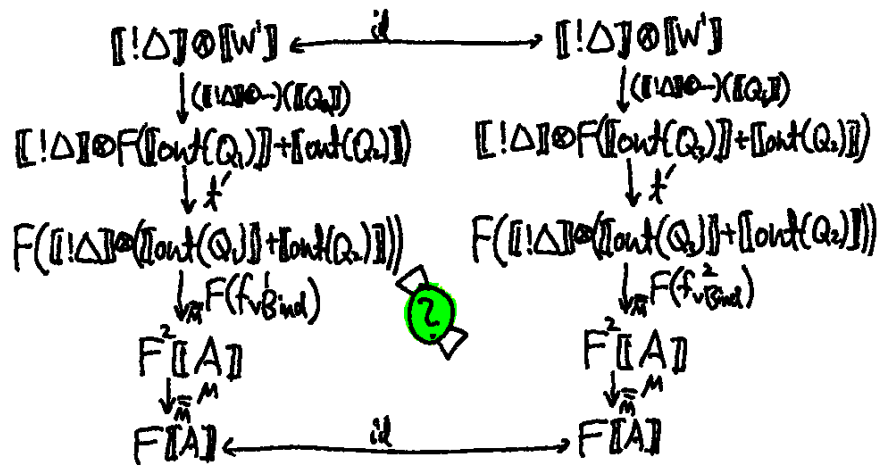
By induction hypothesis:

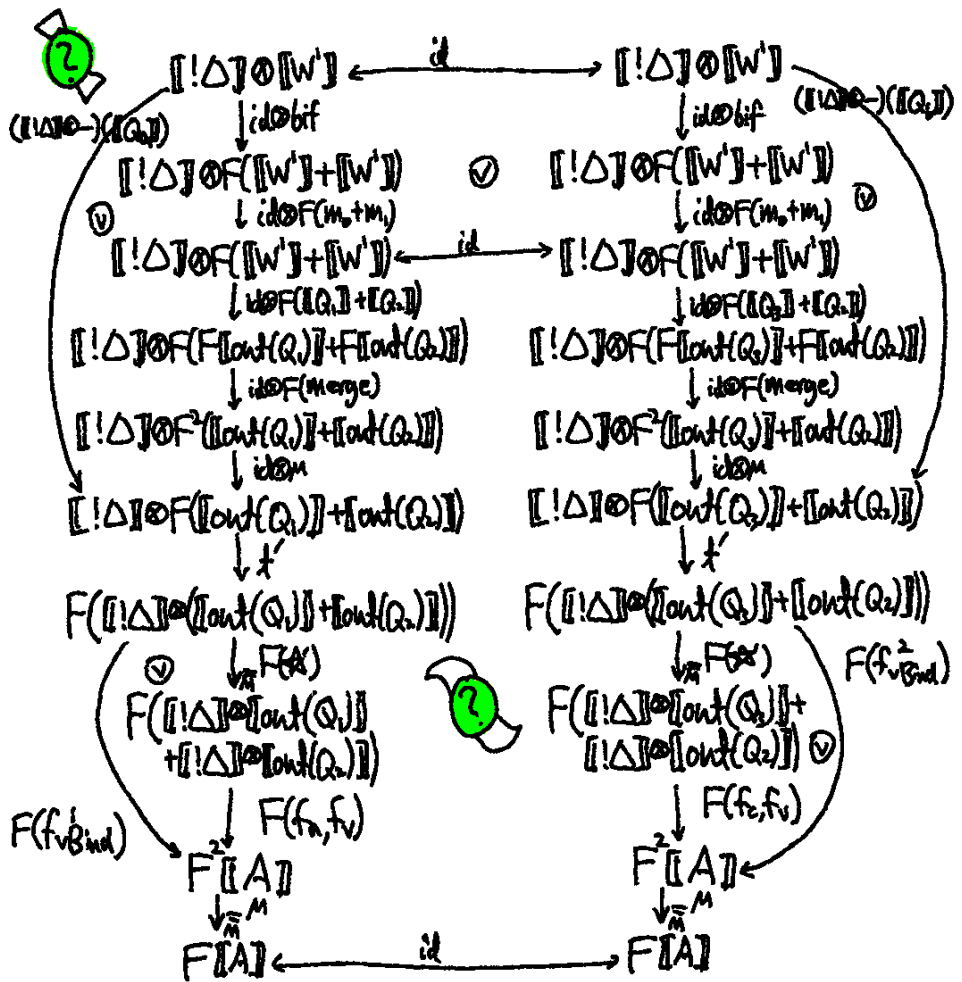
$$\llbracket !\Delta \vdash (Q_1, m_a) : A \rrbracket = \llbracket !\Delta \vdash (Q_3, m_c) : A \rrbracket$$



$$f_a = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q_1), m_a, A) \rrbracket \quad f_c = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q_3), m_c, A) \rrbracket$$

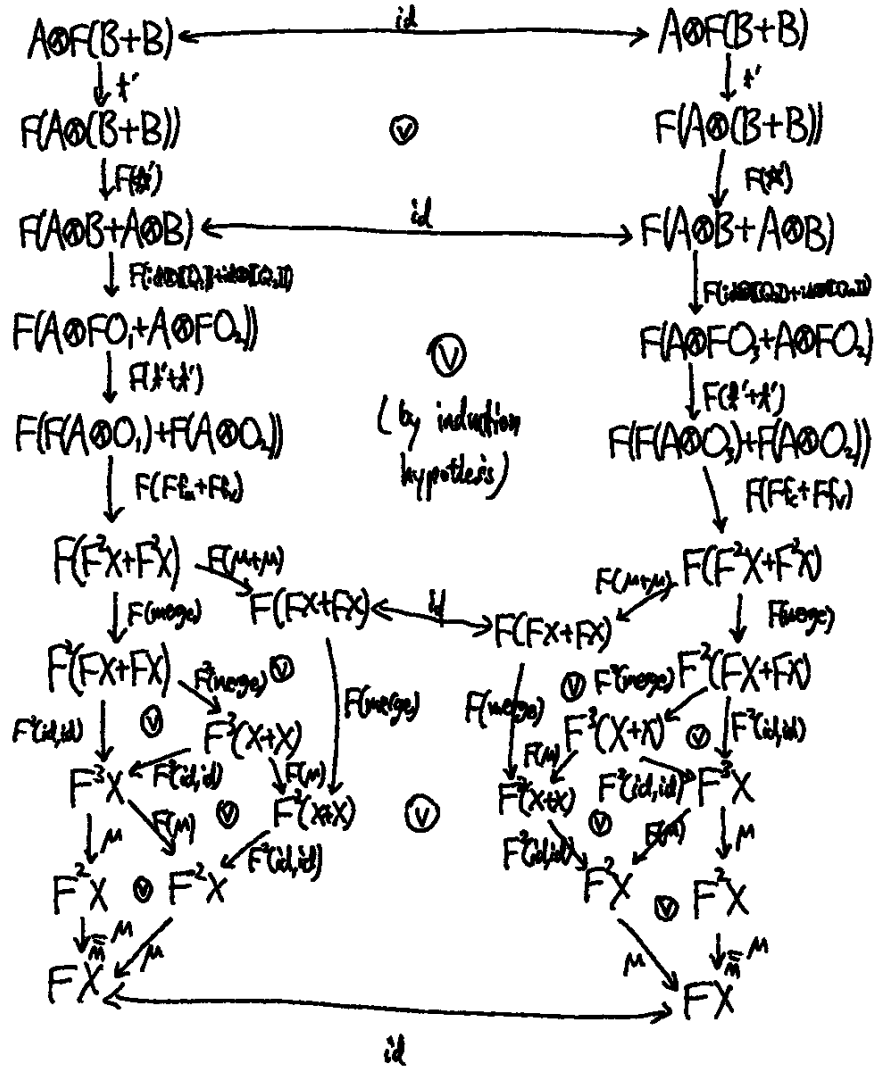
Let $f_v = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q_2), v, A) \rrbracket$, then we show the following:





$$M_2 = [\text{meas}(u, 0)]^*$$

$$M_1 = [\text{meas}(u, 1)]^*$$

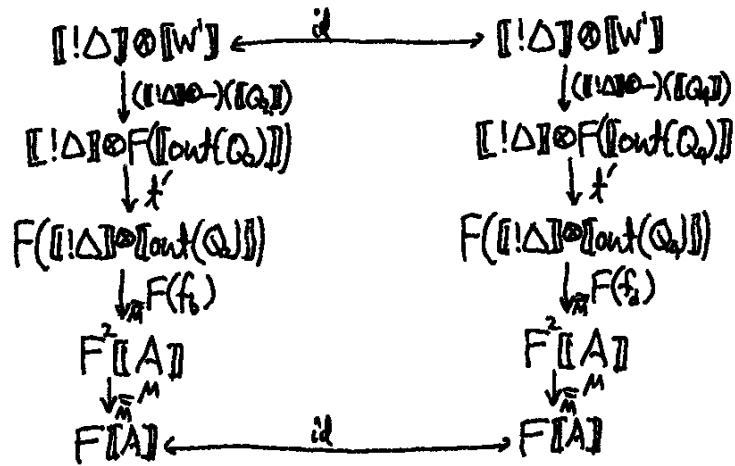


A.15 . Congruence: branching, case 3

$$\begin{array}{c}
 \frac{(Q_2, m_b) \rightarrow (Q_4, m_d)}{(meas\ w\ Q_1\ Q_2), [v, m_b]} \rightarrow (meas\ w\ Q_1\ Q_4), [v, m_d]} \\
 \llbracket !\Delta \vdash (meas\ w\ Q_1\ Q_2), [v, m_b] \rrbracket : A \rrbracket \\
 = \llbracket ([meas\ w\ Q_1\ Q_2], f_{vBind}^1) \rrbracket \\
 = \llbracket !\Delta \rrbracket \otimes \llbracket W^1 \rrbracket \otimes \llbracket [! \Delta] \otimes F([out(Q_1)] + [out(Q_2)]) \rrbracket \\
 \begin{array}{c}
 \downarrow \tilde{x}' \\
 F(\llbracket !\Delta \rrbracket \otimes ([out(Q_1)] + [out(Q_2)])) \\
 \downarrow F(f_{vBind}^1) \\
 F^2 \llbracket A \rrbracket \\
 \downarrow \tilde{m}^M \\
 F \llbracket A \rrbracket
 \end{array} \\
 \begin{array}{l}
 Q_2 = meas\ w\ Q_1\ Q_2 \\
 Q_4 = meas\ w\ Q_1\ Q_4 \\
 W^1 = in(Q_2) = in(Q_1) = in(Q_4) \\
 \quad = in(Q_2) = in(Q_4)
 \end{array} \\
 f_{vBind}^1 = \llbracket vBind(!\Delta, [out(Q_1), out(Q_2)], [v, m_b], A) \rrbracket \\
 \llbracket !\Delta \vdash (meas\ w\ Q_1\ Q_4), [v, m_d] \rrbracket : A \rrbracket \\
 = \llbracket ([meas\ w\ Q_1\ Q_4], f_{vBind}^2) \rrbracket \\
 = \llbracket !\Delta \rrbracket \otimes \llbracket W^2 \rrbracket \otimes \llbracket [! \Delta] \otimes F([out(Q_1)] + [out(Q_4)]) \rrbracket \\
 \begin{array}{c}
 \downarrow \tilde{x}' \\
 F(\llbracket !\Delta \rrbracket \otimes ([out(Q_1)] + [out(Q_4)])) \\
 \downarrow F(f_{vBind}^2) \\
 F^2 \llbracket A \rrbracket \\
 \downarrow \tilde{m}^M \\
 F \llbracket A \rrbracket
 \end{array} \\
 f_{vBind}^2 = \llbracket vBind(!\Delta, [out(Q_1), out(Q_4)], [v, m_d], A) \rrbracket
 \end{array}$$

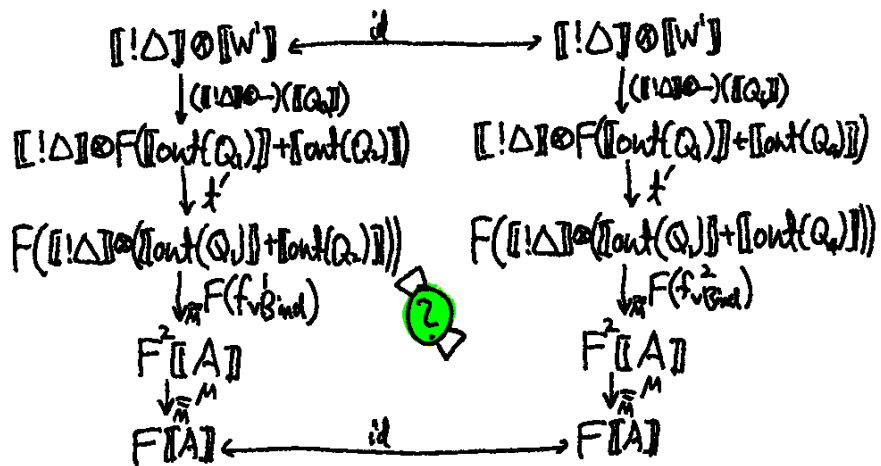
By induction hypothesis:

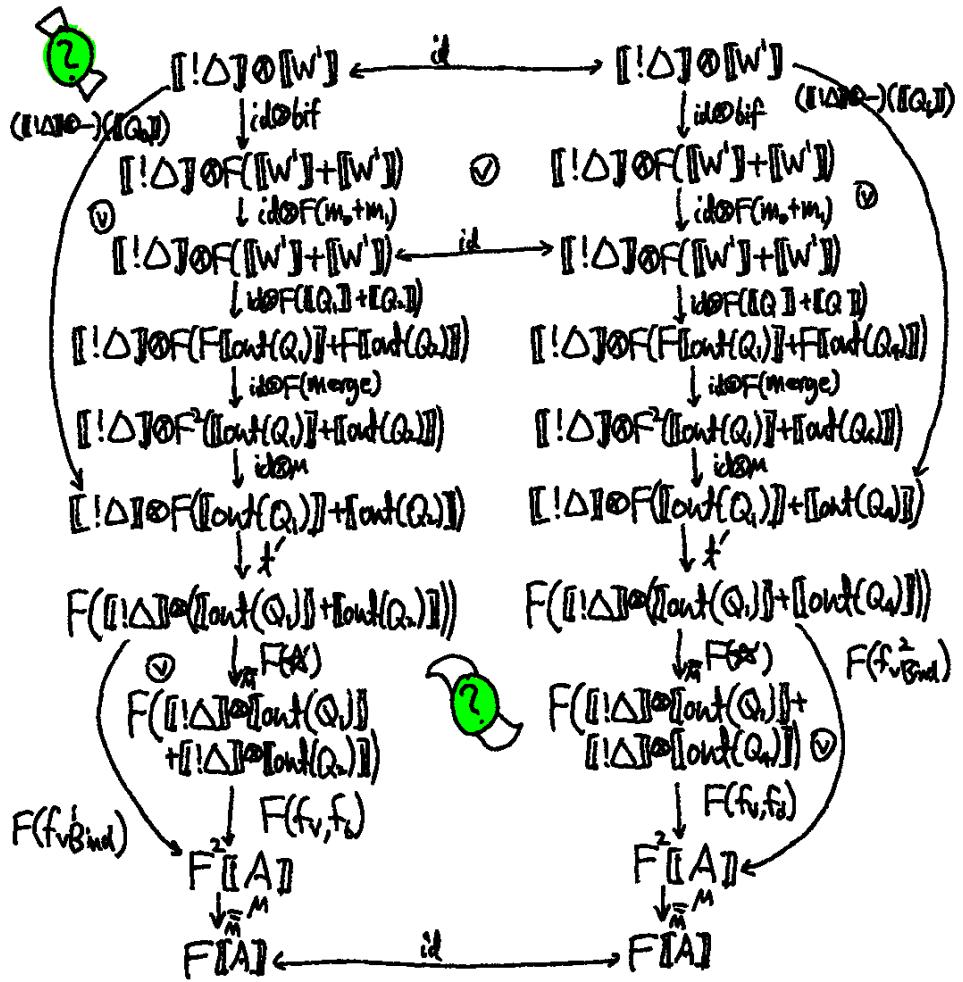
$$\llbracket !\Delta \vdash (Q_2, m_2) : A \rrbracket = \llbracket !\Delta \vdash (Q_1, m_1) : A \rrbracket$$



$$f_2 = \llbracket v \text{Bind}(!\Delta, \text{out}(Q_2), m_2, A) \rrbracket \quad f_1 = \llbracket v \text{Bind}(!\Delta, \text{out}(Q_1), m_1, A) \rrbracket$$

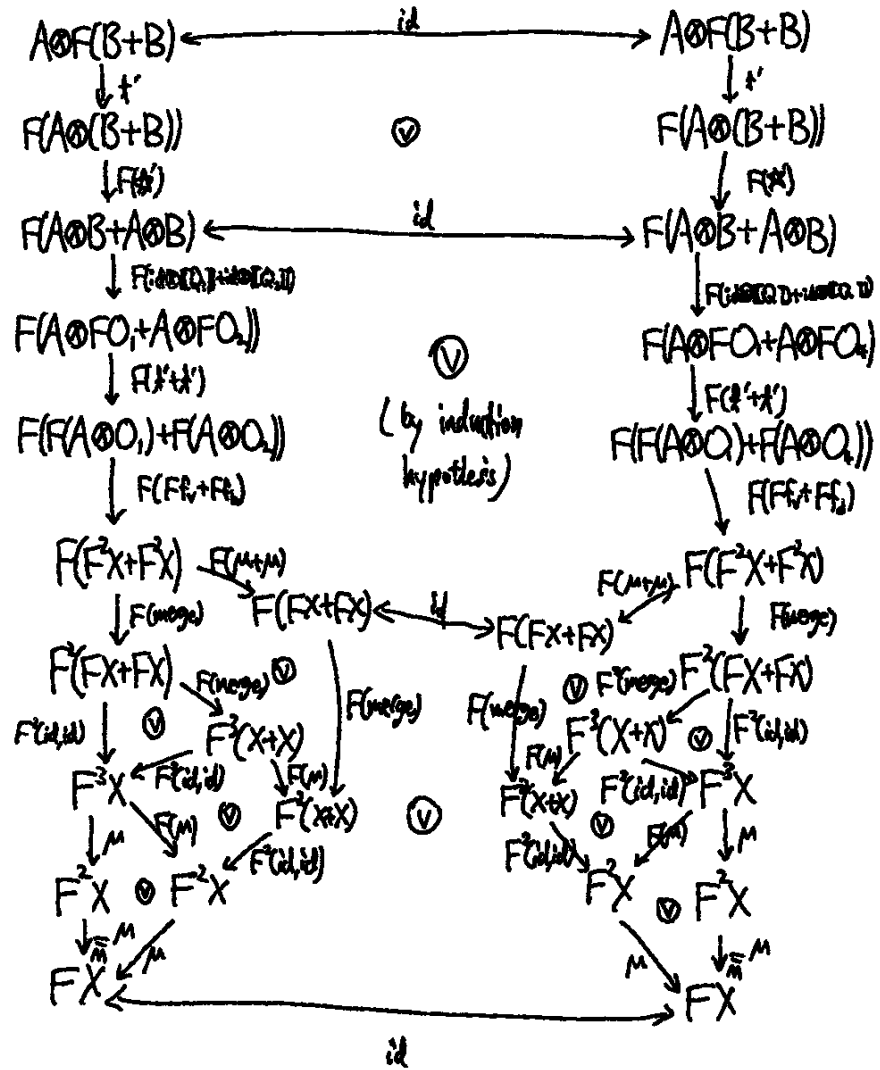
Let $f_v = \llbracket v \text{Bind}(!\Delta, \text{out}(Q_1), v, A) \rrbracket$, then we show the following:





$$M_2 = [\![meas(v, 0)]\!]^*$$

$$M_1 = [\![meas(v, 1)]\!]^*$$



A.16 . Congruence: circuit reduction

$$\frac{(Q_1, m_a) \rightarrow (Q_2, m_b)}{(Q'_1, m_a) \rightarrow (Q'_2, m_b)} \quad \left(\begin{array}{l} Q' = U(W)Q \text{ limit } b \text{ w } Q \\ | \text{ free } w \text{ } Q \end{array} \right)$$

$$\begin{aligned} \llbracket !\Delta \vdash (Q'_1, m_a) : A \rrbracket &= \llbracket ([Q'_1], f^1) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket W^1 \rrbracket \xrightarrow{(\llbracket !\Delta \rrbracket) \otimes (\llbracket Q'_1 \rrbracket)} \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q_1) \rrbracket) \\ &\quad \begin{array}{l} W^1 = \text{in}(Q'_1) \\ \text{out}(Q_1) = \text{out}(Q'_1) \\ f^1 = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q_1), m_a, A) \rrbracket \end{array} \quad \begin{array}{l} \downarrow \text{ } \text{ } \text{ } \\ F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q_1) \rrbracket) \\ \downarrow \text{ } F(f^1) \\ F^2 \llbracket A \rrbracket \\ \downarrow \text{ } \text{ } \text{ } \\ F \llbracket A \rrbracket \end{array} \end{aligned}$$

$$\begin{aligned} \llbracket !\Delta \vdash (Q'_2, m_b) : A \rrbracket &= \llbracket ([Q'_2], f^2) \rrbracket \\ &= \llbracket !\Delta \rrbracket \otimes \llbracket W^2 \rrbracket \xrightarrow{(\llbracket !\Delta \rrbracket) \otimes (\llbracket Q'_2 \rrbracket)} \llbracket !\Delta \rrbracket \otimes F(\llbracket \text{out}(Q_2) \rrbracket) \\ &\quad \begin{array}{l} W^2 = \text{in}(Q'_2) \\ \text{out}(Q_2) = \text{out}(Q'_2) \\ f^2 = \llbracket \nu \text{Bind}(!\Delta, \text{out}(Q_2), m_b, A) \rrbracket \end{array} \quad \begin{array}{l} \downarrow \text{ } \text{ } \text{ } \\ F(\llbracket !\Delta \rrbracket \otimes \llbracket \text{out}(Q_2) \rrbracket) \\ \downarrow \text{ } F(f^2) \\ F^2 \llbracket A \rrbracket \\ \downarrow \text{ } \text{ } \text{ } \\ F \llbracket A \rrbracket \end{array} \end{aligned}$$

By induction hypothesis:

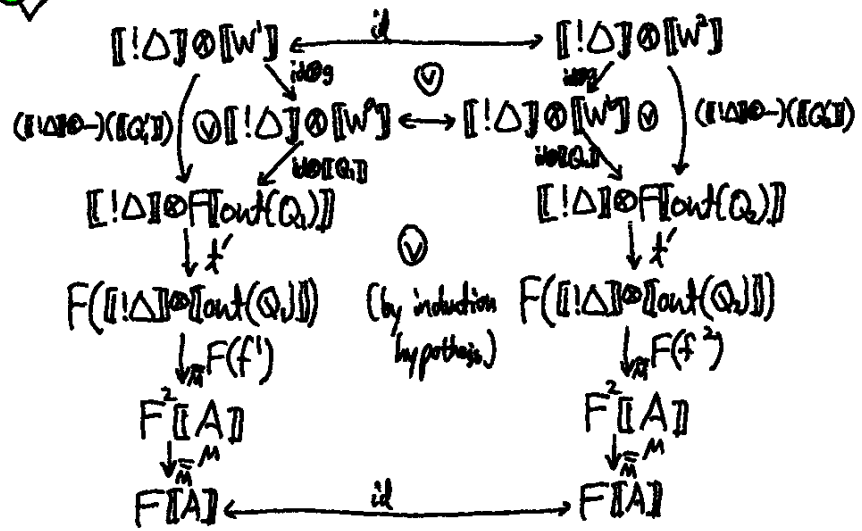
$$[\![\Delta] \vdash (Q_1, m_1) : A]\!] = [\![\Delta] \vdash (Q_2, m_2) : A]\!]$$

$$\begin{array}{ccc}
 [\![\Delta] \vdash [W']]\!] & \xleftarrow{id} & [\![\Delta] \vdash [W']]\!] \\
 \downarrow (\text{in } \text{in} \rightarrow) \text{in } (Q_1) & & \downarrow (\text{in } \text{in} \rightarrow) \text{in } (Q_2) \\
 [\![\Delta] \vdash F(\text{out}(Q_1))]\!] & & [\![\Delta] \vdash F(\text{out}(Q_2))]\!] \\
 \downarrow f' & & \downarrow f' \\
 F([\![\Delta] \vdash \text{out}(Q_1)]\!]) & & F([\![\Delta] \vdash \text{out}(Q_2)]\!]) \\
 \downarrow_{\text{in}} F(f') & & \downarrow_{\text{in}} F(f') \\
 F^2[A] & & F^2[A] \quad \left(\begin{array}{l} W^* = \text{in}(Q_1) \\ W^* = \text{in}(Q_2) \end{array} \right) \\
 \downarrow_{\text{in}}^M & & \downarrow_{\text{in}}^M \\
 F[A] & \xleftarrow{id} & F[A]
 \end{array}$$

Then, we show the following!

$$\begin{array}{ccc}
 [\![\Delta] \vdash [W']]\!] & \xleftarrow{id} & [\![\Delta] \vdash [W']]\!] \\
 \downarrow (\text{in } \text{in} \rightarrow) \text{in } (Q_1) & & \downarrow (\text{in } \text{in} \rightarrow) \text{in } (Q_2) \\
 [\![\Delta] \vdash F(\text{out}(Q_1))]\!] & & [\![\Delta] \vdash F(\text{out}(Q_2))]\!] \\
 \downarrow f' & & \downarrow f' \\
 F([\![\Delta] \vdash \text{out}(Q_1)]\!]) & & F([\![\Delta] \vdash \text{out}(Q_2)]\!]) \\
 \downarrow_{\text{in}} F(f') & & \downarrow_{\text{in}} F(f') \\
 F^2[A] & & F^2[A] \\
 \downarrow_{\text{in}}^M & & \downarrow_{\text{in}}^M \\
 F[A] & \xleftarrow{id} & F[A]
 \end{array}$$

?



where $g = [U(w)]^\circ, [\text{init}(h, w)]^\circ, [\text{free}(w)]^\circ$

B - Synthèse de la thèse

B.1 . Contexte

Avec ses succès récents en développement des ordinateurs quantiques, la recherche en informatique quantique a gagné l'intérêt de celles et ceux qui veulent utiliser la puissance de ce nouveau modèle de calcul basé sur la mécanique quantique. Pourtant, il est difficile de concevoir des algorithmes quantiques et de vérifier qu'ils fonctionnent de la manière souhaitée à cause de la difficulté du raisonnement sur ces algorithmes qui souvent nécessitent l'analyse de propriétés sur l'état quantique. L'état quantique est formalisé par un vecteur dans un espace de Hilbert de taille exponentielle par rapport au nombre de qubits. La recherche en langage de programmation pour l'ordinateur quantique cherche à résoudre le problème de la description sur les algorithmes quantiques et leurs propriétés pour nous aider à raisonner sur les algorithmes quantiques de manière logique, basé sur le modèle formel du calcul quantique.

Le modèle Qram est un modèle de calcul quantique pratique composé d'un ordinateur classique et un processus quantique qui communiquent entre eux. Le programme est exécuté sur l'ordinateur classique. Il envoie les instructions correspondant aux opérateurs quantiques sur le co-processeur, et reçoit le résultat de l'observation de l'état quantique. Ce modèle est considéré comme un modèle standard et plusieurs langages de programmation ont été conçus en basant sur ce modèle.

B.2 . Le Problème

Alors que les programmes dans ce modèle sont capables de réaliser tout calcul quantique grâce à l'usage de la mémoire quantique, il est difficile de les analyser sans l'aide d'un autre ordinateur quantique. Ce problème suscite le besoin pour des méthodes formelles pour les langages programmations quantiques : les outils formels pour raisonner sur l'optimisation du code, pour l'analyser des ressources, et pour spécifier et prouver les propriétés des programmes quantiques.

La sémantique catégorique fait partie de ces méthodes qui fait le lien entre les opérateurs quantiques et les programmes et introduit le système logique qui peut décrire les propriétés sur l'état quantique dans le système de types. Bien que plusieurs travaux proposent des sémantiques catégoriques pour les langages de description de circuits quantiques, aucun ne supporte l'usage du résultat d'une mesure au sein du processeur classique (le "levage dynamique").

B.3 . Solution Proposée

Dans cette thèse, nous formalisons le levage dynamique qui transfère le résultat d'observations sur l'état quantique à l'information classique dans un langage de programmation de description de circuit quantique. En suivant l'approche du langage Proto-Quipper, nous définissons un langage typé de description de circuit quantique où l'information quantique levée est incorporée dans la structure ramifiée. Ensuite, le levage dynamique est formalisé dans le cadre de la sémantique opérationnelle et la sémantique catégorique.

Notre sémantique catégorique est basée sur le modèle de Francisco Rios et Peter Selinger pour le langage programmation Proto-Quipper-M. Pourtant, pour formaliser le levage dynamique, nous construisons une catégorie de Kleisli en capturant la mesure quantique comme un effet de bord sur une catégorie concrète pour le circuit avec la mesure quantique. Nous prouvons le théorème de correction de la sémantique catégorique par rapport à la sémantique opérationnelle.

B.4 . Discussion et Ouverture

Le langage que nous avons proposé nous aide à décrire les programmes basés sur le modèle Qram. Les circuits quantiques sont construits par le calcul classique et transféré au coprocesseur en utilisant des opérateurs comme `box` et `unbox`. Ensuite, la mesure crée un ensemble de calculs qui dépendent du résultat de la mesure. Le système de types s'assure que tous les programmes typés peuvent être réalisés physiquement puisqu'ils ne violent pas le no-cloning théorème.

Dans notre système de types, la validité du programme est définie par le jugement de type. Les types peuvent être considérés comme les ensembles de programmes valides qui satisfont certaines propriétés qui correspondent au type. Pourtant, on pourrait essayer de définir et vérifier les propriétés fonctionnelles de programmes en utilisant le système de types.

Dans ce but, il est nécessaire d'enrichir l'exprimabilité du système du type. Pour le faire, il y a trois traits de types qui seraient utiles. Premièrement, les types qui désignent les sous-ensembles ou les sous-espaces de l'espace quantique nous aideront à dénoter la fonction précise du programme. Deuxièmement, on pourrait définir les programmes paramétrés avec les types dépendants. Troisièmement, la relation d'égalité de programmes est nécessaire pour définir la théorie des états quantiques. Il reste pour la recherche en avenir à formaliser le système de types consistant qui satisfait les requirements.