

A Value Trick for Modal Type Systems

ANTON LORENZEN, University of Edinburgh, United Kingdom

WENHAO TANG, University of Edinburgh, United Kingdom

SAM LINDLEY, University of Edinburgh, United Kingdom

Modal type systems are a powerful tool for tracking properties in programming languages. They have long been used to track properties of *computations*, such as mobility, co-effects or erasability. In recent years, several authors have used them to track properties of *values*, such as stack location, linearity, purity or acyclicity. However, to support the latter applications, one needs to restrict modal type systems appropriately. In this abstract, we discuss the *value trick* as a unifying principle connecting several approaches from the literature. This is a work-in-progress, based on earlier work of the authors [18, 28].

Modal type systems. Modal logics have become widely used in the field of programming languages. Under the propositions-as-types correspondence, the type $\Box A$ denotes terms of type A that additionally satisfy some property. Usually, one assumes the axioms of IS4: that the property can be forgotten and duplicated. Most modal type systems are presented either in dual-context style [25] or, more recently, in Fitch-style [8]. In this work, we use the latter; stripping away the usual type formers (lambdas, products, sums), the relevant rules are:

$$\frac{\mathfrak{L} \notin \Gamma'}{\Gamma, x : A, \Gamma' \vdash x : A} \qquad \frac{\Gamma, \mathfrak{L} \vdash e : A}{\Gamma \vdash \text{box } e : \Box A} \qquad \frac{\Gamma \vdash e : \Box A}{\Gamma, \Gamma' \vdash \text{unbox } e : A}$$

Call-by-Name Interpretation. Terms are endowed with a reduction relation, which evaluates a term to a value. For modal type systems, it is common to use call-by-name interpretation. For elimination rules like $\text{unbox } e$, the inner term e is evaluated until a suitable value $\text{box } e$ is reached. In contrast, the term in the box is not evaluated directly and is returned by the elimination rule:

$$\text{unbox } (\text{box } e) \rightsquigarrow e \qquad E ::= [\cdot] \mid \text{unbox } E$$

Such semantics have been used to track properties of computations such as mobility [19], co-effects [24], erasability [1, 4], usage in call-by-name [1, 6, 20], and staged computation [10]. However, as was already remarked by Murphy et al. [19], the key property to make this sound is that "we do not attempt to evaluate under the box".

Call-by-Value Interpretation. In recent years, it has become fashionable to use modal type systems also for languages with call-by-value semantics. In this setting, a natural design choice is to allow evaluation under the box:

$$\text{unbox } (\text{box } v) \rightsquigarrow v \qquad E ::= [\cdot] \mid \text{box } E \mid \text{unbox } E$$

When evaluation under the box is allowed, the modality no longer tracks properties of the computation, but rather of the returned value v . Call-by-value semantics has been used for modal type systems to track properties such as temporality [2], stack allocation [11, 18], affinity [18], purity [7], acyclicity [15], and effects [27, 28].

However, the above modal type system is not sound for these applications. As IS4 is a normal modal logic, the calculus admits the K axiom ($\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$). However, this axiom does not hold if the calculus contains primitives that can generate certain new values out of thin air. For example, if $\Box A$ denotes non-linear values of a linear type A , then $\Box \mathbb{1}$ and $\Box(\mathbb{1} \rightarrow \text{File})$ are inhabited by unit and `openFile`, but $\Box \text{File}$ should not be inhabited, as files should be kept linear.

The Continuation Trick. Wadler [30] solves this problem by using a continuation-passing style (CPS) interface for primitives that generate new values, like $\text{openFile} : (\text{File} \rightarrow \text{File} \times X) \rightarrow X$. The CPS interface ensures that no function returns a linear value out of thin air. This trick has since been used by several authors [3, 7].

The Value Trick. In this work, we propose a simpler solution, that does not require a CPS interface. Instead, we restrict the box and unbox rules for the modality to values:

$$\frac{\mathfrak{L} \notin \Gamma'}{\Gamma, x : A, \Gamma' \vdash x : A'} \quad \frac{\Gamma, \mathfrak{L} \vdash v : A}{\Gamma \vdash \text{box } v : \Box A} \quad \frac{\Gamma \vdash v : \Box A}{\Gamma, \Gamma' \vdash \text{unbox } v : A}$$

Furthermore, both $\text{box } v$ and $\text{unbox } v$ are part of the syntax of values, which makes it possible to unbox a variable under a box. While $\text{unbox } (\text{box } v)$ is a redex, the reduction has no side-effects or cost at runtime, which makes it possible to perform it at any time, similar to "complex values" [16].

Let-style Unbox in Fine-grained CBV. We can generalise the value trick to multimodal type theory (MTT) [13] in fine-grain call-by-value style [17]. Variants of this calculus were used by Ahman [2], Tang et al. [28] and Tang and Lindley [27]. MTT is parameterised by a 2-category of modes and modalities. For simplicity, we restrict ourselves to one mode. In this setting, variables in the context, locks, and boxes are annotated by a monoid of modalities μ, ν which can be composed using \circ . We write $\text{locks}(\Gamma')$ for the composition of the modalities of the locks in Γ' . The relevant rules are:

$$\frac{\mu \Rightarrow \text{locks}(\Gamma')}{\Gamma, x :_{\mu} A, \Gamma' \vdash x : A} \quad \frac{\Gamma, \mathfrak{L}_{\mu} \vdash v : A}{\Gamma \vdash \text{box}_{\mu} v : \Box_{\mu} A} \quad \frac{\Gamma, \mathfrak{L}_{\nu} \vdash v : \Box_{\mu} A \quad \Gamma, x :_{\nu \circ \mu} A \vdash e : B}{\Gamma \vdash \text{let}_{\nu} \text{box}_{\mu} x = v \text{ in } e : B}$$

The main change from MTT is that the box and unbox rules are restricted to values v . The intuition here is that whenever a precondition introduces a lock to the context, the term in it has to be a value (or a computation that is suspended by the semantics).

Graded Call-by-Push-Value. Torczon et al. [29] propose an elegant formulation of graded [1, 6] type theory in call-by-push-value [16] where the grades attach to values. Graded type theory is parameterised by a semiring of grades q , which can be composed using $+$ and \cdot . We write $q \cdot \Gamma$ for the context that arises by multiplying all grades in Γ by q and $\Gamma_1 + \Gamma_2$ for the context that arises by adding the grades of each variable in Γ_1 and Γ_2 . We can express this using the rules:

$$\frac{}{0 \cdot \Gamma, x :^1 A, 0 \cdot \Gamma' \vdash x : A} \quad \frac{\Gamma \vdash v : A}{q \cdot \Gamma \vdash \text{box}_q v : \Box^q A} \quad \frac{\Gamma_1 \vdash v : \Box^{q_2} A \quad \Gamma_2, x :^{q_1 \cdot q_2} A \vdash c : \underline{B}}{q_1 \cdot \Gamma_1 + \Gamma_2 \vdash \text{let}_{q_1} \text{box}_{q_2} x = v \text{ in } c : \underline{B}}$$

Comparing this box rule to MTT, we can see that graded type theory multiplies the context in the conclusion, while MTT puts a lock into the context in the premise. Again, we have the intuition that whenever the context arising from a precondition is multiplied by a grade, then the term in the precondition has to be a value. However, Torczon et al. [29] actually do not follow this completely as they do not restrict the elimination rule to values (COEFF-LETIN). We believe that the issue that they identify in Section 3.2 of their paper arises from this choice and could be remedied by restricting the elimination rule to values as well.

Mode Qualifiers. In most modal type systems, the modality commutes with products and sums. This makes the syntax of value-restricted modal type systems redundant, as the modality can be pushed down into the type: for example, it should not matter whether we write $\Box(A, B)$ or $(\Box A, \Box B)$. This motivates the calculus of mode qualifiers [18] (related to the earlier work of [11, 12, 21, 22]),

where we push the modality as far down as possible, until we reach either a variable or a lambda abstraction (through which we can not push the modality).

$$\frac{\mu_1 \leq \mu_2}{\Gamma, x : A @ \mu_1 \vdash x : A @ \mu_2} \quad \frac{\Gamma, \mu, x : A @ \mu_1 \vdash e : B @ \mu_2}{\Gamma \vdash \lambda x. e : (A @ \mu_1 \rightarrow B @ \mu_2) @ \mu} \quad \frac{\Gamma \vdash v : A @ \nu \circ \mu}{\Gamma \vdash \text{box}_\nu v : \Box^\nu A @ \mu}$$

Adjoint Natural Deduction. Jang et al. [14] define a deduction system for tracking properties of values. They make the observation that their system models the usual IS4 modality as $\Box_{\text{IS4}} A = \Box(\mathbb{1} \rightarrow A)$. This suggests that in order to track properties of computations, we can track properties of closure values instead.

Correspondence. In the Appendix, we sketch a correspondence between the five approaches above when restricted to a single property. This shows that they are all equally useful for tracking simple properties of values and that the one should pick a type system based on the other features it supports (like multi-modalities, substructural tracking, or type inference).

Which modal logic is this? In the appendix, we derive terms for $T : \Box A \rightarrow A$, $4 : \Box A \rightarrow \Box \Box A$, $X : \Box(A \rightarrow B) \times \Box(B \rightarrow C) \rightarrow \Box(A \rightarrow C)$, and $N : \Box \mathbb{1}$. Our calculus does not admit the rule of monotonicity ($\vdash A \rightarrow B$ implies $\vdash \Box A \rightarrow \Box B$), the K Axiom ($\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$), or the rule of congruence ($\vdash A \leftrightarrow B$ implies $\vdash \Box A \leftrightarrow \Box B$). While the rule of necessitation ($\vdash A$ implies $\vdash \Box A$) is admissible in the base calculus, in practice one often adds primitives that refute this rule. At the workshop, we hope to explore possible connections to hyperintensional logics [5, 9, 23, 26].

Acknowledgments

We thank Nachi Valliappan and Ian Shillito for discussions on modal logic. Our intuition for the value trick was shaped by previous discussions with Leo White and Stephen Dolan. Sam Lindley was supported by UKRI Future Leaders Fellowship “Effect Handler Oriented Programming” (MR/T043830/1 and MR/Z000351/1).

References

- [1] Andreas Abel and Jean-Philippe Bernardy. 2020. A unified view of modalities in type systems. *Proceedings of the ACM on Programming Languages* 4, ICFP (2020), 1–28. <https://doi.org/10.1145/3408972>
- [2] Danel Ahman. 2023. When programs have to watch paint dry. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 1–23.
- [3] Michael Arntzenius and Neel Krishnaswami. 2019. Seminaïve evaluation for a higher-order functional language. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–28.
- [4] Robert Atkey. 2018. Syntax and semantics of quantitative type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 56–65. <https://doi.org/10.1145/3209108.3209189>
- [5] Francesco Berto and Daniel Nolan. 2021. Hyperintensionality. (2021).
- [6] Pritam Choudhury, Harley Eades III, Richard A Eisenberg, and Stephanie Weirich. 2021. A graded dependent type system with a usage-aware semantics. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–32. <https://doi.org/10.1145/3434331>
- [7] Vikraman Choudhury and Neel Krishnaswami. 2020. Recovering purity with comonads and capabilities. *Proceedings of the ACM on Programming Languages* 4, ICFP (2020), 1–28.
- [8] Randal Clouston. 2018. Fitch-style modal lambda calculi. In *International Conference on Foundations of Software Science and Computation Structures*. Springer, 258–275.
- [9] Max J Cresswell. 1975. Hyperintensional logic. *Studia Logica: An International Journal for Symbolic Logic* 34, 1 (1975), 25–38.
- [10] Rowan Davies and Frank Pfenning. 1996. A Modal Analysis of Staged Computation. In *Conference Record of POPL’96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21–24, 1996*, Hans-Juergen Boehm and Guy L. Steele Jr. (Eds.). ACM Press, 258–270. <https://doi.org/10.1145/237721.237788>
- [11] Stephen Dolan and Leo White. 2022. Stack allocation for OCaml. Talk. In *OCaml workshop 2022*. ICFP.

- [12] Jeffrey S Foster, Manuel Fähndrich, and Alexander Aiken. 1999. A theory of type qualifiers. *ACM SIGPLAN Notices* 34, 5 (1999), 192–203. <https://doi.org/10.1145/301618.301665>
- [13] Daniel Gratzer, GA Kavvos, Andreas Nuyts, and Lars Birkedal. 2020. Multimodal dependent type theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. 492–506. <https://doi.org/10.1145/3373718.3394736>
- [14] Junyoung Jang, Sophia Roshal, Frank Pfenning, and Brigitte Pientka. 2024. Adjoint natural deduction. In *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 15–1.
- [15] Paulette Koronkevich and William J Bowman. 2025. Type Universes as Kripke Worlds. *Proceedings of the ACM on Programming Languages* 9, ICFP (2025), 784–813.
- [16] Paul Blain Levy. 1999. Call-by-push-value: A subsuming paradigm. In *International Conference on Typed Lambda Calculi and Applications*. Springer, 228–243.
- [17] Paul Blain Levy, John Power, and Hayo Thielecke. 2003. Modelling environments in call-by-value programming languages. *Inf. Comput.* 185, 2 (2003), 182–210. [https://doi.org/10.1016/S0890-5401\(03\)00088-9](https://doi.org/10.1016/S0890-5401(03)00088-9)
- [18] Anton Lorenzen, Leo White, Stephen Dolan, Richard A Eisenberg, and Sam Lindley. 2024. Oxidizing OCaml with modal memory management. *Proceedings of the ACM on Programming Languages* 8, ICFP (2024), 485–514.
- [19] Tom Murphy, Karl Crary, Robert Harper, and Frank Pfenning. 2004. A symmetric modal lambda calculus for distributed computing. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science, 2004*. IEEE, 286–295.
- [20] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. 2019. Quantitative program reasoning with graded modal types. *Proceedings of the ACM on Programming Languages* 3, ICFP (2019), 1–30. <https://doi.org/10.1145/3341714>
- [21] Leo Osvald, Grégory Essertel, Xilun Wu, Lilliam I González Alayón, and Tiark Rompf. 2016. Gentrification gone too far? affordable 2nd-class values for fun and (co-) effect. *ACM SIGPLAN Notices* 51, 10 (2016), 234–251.
- [22] Leo Osvald and Tiark Rompf. 2017. Rust-like borrowing with 2nd-class values (short paper). In *Proceedings of the 8th ACM SIGPLAN International Symposium on Scala*. 13–17.
- [23] Matteo Pascucci and Igor Sedlár. 2025. Hyperintensional models for non-congruential modal logics. *Logic Journal of the IGPL* 33, 5 (2025).
- [24] Tomas Petricek, Dominic Orchard, and Alan Mycroft. 2014. Coeffects: A Calculus of Context-dependent Computation. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming (Gothenburg, Sweden) (ICFP '14)*. ACM, 123–135. <https://doi.org/10.1145/2628136.2628160>
- [25] Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Mathematical structures in computer science* 11, 4 (2001), 511–540.
- [26] Igor Sedlár. 2021. Hyperintensional logics for everyone. *Synthese* 198, 2 (2021), 933–956.
- [27] Wenhao Tang and Sam Lindley. 2026. Rows and Capabilities as Modal Effects. *Proceedings of the ACM on Programming Languages* 10, POPL (2026), 923–950.
- [28] Wenhao Tang, Leo White, Stephen Dolan, Daniel Hillerström, Sam Lindley, and Anton Lorenzen. 2025. Modal effect types. *Proceedings of the ACM on Programming Languages* 9, OOPSLA1 (2025), 1130–1157.
- [29] Cassia Torczon, Emmanuel Suárez Acevedo, Shubh Agrawal, Joey Velez-Ginorio, and Stephanie Weirich. 2024. Effects and coeffects in call-by-push-value. *Proceedings of the ACM on Programming Languages* 8, OOPSLA2 (2024), 1108–1134.
- [30] Philip Wadler. 1993. A taste of linear logic. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 185–210.

Claude Opus was utilised for significant parts of this appendix, including text, typing rules, semantics and proofs. All output was carefully checked and corrected by the authors.

A Calculi

A.1 Fitch-style IS4

Syntax.

$$\begin{aligned}
 A, B &::= A \rightarrow B \mid \Box A \mid A \times B \mid A + B \mid \mathbb{1} \\
 v &::= x \mid \lambda x. e \mid \text{box } v \mid \text{unbox } v \mid (v_1, v_2) \mid () \mid \text{inl } v \mid \text{inr } v \\
 e &::= v \mid e_1 e_2 \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{dist}^\times e \mid \text{let}(x, y) = e_1 \text{ in } e_2 \mid \text{dist}^+ e \mid \text{case } e \{x.e_1; y.e_2\} \\
 \Gamma &::= \cdot \mid \Gamma, x : A \mid \Gamma, \mathfrak{A}
 \end{aligned}$$

Typing rules.

$$\boxed{\Gamma \vdash e : A}$$

$$\begin{array}{c}
 \frac{\mathfrak{A} \notin \Gamma'}{\Gamma, x : A, \Gamma' \vdash x : A} \text{VAR} \qquad \frac{\Gamma, x : A \vdash e : B}{\Gamma \vdash \lambda x. e : A \rightarrow B} \text{LAM} \qquad \frac{\Gamma \vdash e_1 : A \rightarrow B \quad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1 e_2 : B} \text{APP} \\
 \\
 \frac{\Gamma, \mathfrak{A} \vdash v : A}{\Gamma \vdash \text{box } v : \Box A} \Box\text{I} \qquad \frac{\Gamma \vdash v : \Box A}{\Gamma, \Gamma' \vdash \text{unbox } v : A} \Box\text{E} \qquad \frac{}{\Gamma \vdash () : \mathbb{1}} \text{UNIT} \\
 \\
 \frac{\Gamma \vdash e_1 : A \quad \Gamma, x : A \vdash e_2 : B}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : B} \text{LET} \qquad \frac{\Gamma \vdash e : \Box(A \times B)}{\Gamma \vdash \text{dist}^\times e : \Box A \times \Box B} \text{DIST}^\times \\
 \\
 \frac{\Gamma \vdash v_1 : A \quad \Gamma \vdash v_2 : B}{\Gamma \vdash (v_1, v_2) : A \times B} \text{PAIR} \qquad \frac{\Gamma \vdash e_1 : A \times B \quad \Gamma, x : A, y : B \vdash e_2 : C}{\Gamma \vdash \text{let}(x, y) = e_1 \text{ in } e_2 : C} \text{SPLIT} \\
 \\
 \frac{\Gamma \vdash v : A}{\Gamma \vdash \text{inl } v : A + B} \text{INL} \qquad \frac{\Gamma \vdash v : B}{\Gamma \vdash \text{inr } v : A + B} \text{INR} \qquad \frac{\Gamma \vdash e : \Box(A + B)}{\Gamma \vdash \text{dist}^+ e : \Box A + \Box B} \text{DIST}^+ \\
 \\
 \frac{\Gamma \vdash e : A + B \quad \Gamma, x : A \vdash e_1 : C \quad \Gamma, y : B \vdash e_2 : C}{\Gamma \vdash \text{case } e \{x.e_1; y.e_2\} : C} \text{CASE}
 \end{array}$$

Call-by-value semantics.

$$\boxed{e \rightsquigarrow e'}$$

$$\begin{aligned}
& (\lambda x. e) v \rightsquigarrow e[v/x] \\
& \text{let } x = v \text{ in } e \rightsquigarrow e[v/x] \\
& \text{unbox } (\text{box } v) \rightsquigarrow v \\
& \text{let } (x, y) = (v_1, v_2) \text{ in } e \rightsquigarrow e[v_1/x, v_2/y] \\
& \text{dist}^\times (\text{box } (v_1, v_2)) \rightsquigarrow (\text{box } v_1, \text{box } v_2) \\
& \text{dist}^+ (\text{box } (\text{inl } v)) \rightsquigarrow \text{inl } (\text{box } v) \\
& \text{dist}^+ (\text{box } (\text{inr } v)) \rightsquigarrow \text{inr } (\text{box } v) \\
& \text{case } (\text{inl } v) \{x.e_1; y.e_2\} \rightsquigarrow e_1[v/x] \\
& \text{case } (\text{inr } v) \{x.e_1; y.e_2\} \rightsquigarrow e_2[v/y]
\end{aligned}$$

Evaluation contexts.

$$\begin{aligned}
E ::= [\cdot] \mid E e_2 \mid v_1 E \mid \text{dist}^\times E \mid \text{dist}^+ E \\
\mid \text{let } x = E \text{ in } e \mid \text{let } (x, y) = E \text{ in } e \mid \text{case } E \{x.e_1; y.e_2\}
\end{aligned}
\qquad
\frac{e \rightsquigarrow e'}{E[e] \rightsquigarrow E[e']} \text{CTX}$$

Parameterisation. None

Derivations. We derive terms for the following types:

$$\begin{aligned}
T &= \lambda x. \text{unbox } x : \Box A \rightarrow A \\
4 &= \lambda x. \text{box } (\text{box } (\text{unbox } x)) : \Box A \rightarrow \Box \Box A \\
N &= \text{box } () : \Box \mathbb{1} \\
X &= \lambda p. \text{let } (f, g) = p \text{ in } \text{box } (\lambda x. (\text{unbox } g) ((\text{unbox } f) x)) \\
&\quad : \Box(A \rightarrow B) \times \Box(B \rightarrow C) \rightarrow \Box(A \rightarrow C) \\
M^\times &= \lambda p. \text{let } (x, y) = p \text{ in } \text{box } (\text{unbox } x, \text{unbox } y) : \Box A \times \Box B \rightarrow \Box(A \times B) \\
M^+ &= \lambda s. \text{case } s \{x. \text{box } (\text{inl } (\text{unbox } x)); y. \text{box } (\text{inr } (\text{unbox } y))\} : \Box A + \Box B \rightarrow \Box(A + B)
\end{aligned}$$

The rule of necessitation ($\vdash A$ implies $\vdash \Box A$) is admissible: if $\vdash e : A$, then e evaluates to a closed value v with $\vdash v : A$ by type safety, and since v has no free variables, $\mathfrak{L} \vdash v : A$, so $\vdash \text{box } v : \Box A$.

A.2 Let-style Unbox in Fine-grained CBV

Syntax.

$$\begin{aligned}
A, B ::= A^\mu \rightarrow B \mid \Box_\mu A \mid A \times B \mid A + B \mid \mathbb{1} \\
v ::= x \mid \lambda x. e \mid \text{box}_\mu v \mid (v_1, v_2) \mid () \mid \text{inl } v \mid \text{inr } v \\
e ::= v \mid e v \mid \text{let } x = e_1 \text{ in } e_2 \mid \text{let}_\mu \text{box}_\nu x = v \text{ in } e \\
\mid \text{let}_\mu (x, y) = v \text{ in } e \mid \text{case}_\mu v \{x.e_1; y.e_2\} \\
\Gamma ::= \cdot \mid \Gamma, x ;_\mu A \mid \Gamma, \mathfrak{L}_\mu
\end{aligned}$$

We define $\text{locks}(\Gamma)$ as the composition of all locks in Γ :

$$\begin{aligned}
\text{locks}(\cdot) &= \text{id} \\
\text{locks}(\Gamma, x ;_\mu A) &= \text{locks}(\Gamma) \\
\text{locks}(\Gamma, \mathfrak{L}_\mu) &= \mu \circ \text{locks}(\Gamma)
\end{aligned}$$

Typing rules.

$\boxed{\Gamma \vdash e : A}$

$$\begin{array}{c}
\frac{\mu \Rightarrow \text{locks}(\Gamma')}{\Gamma, x :_{\mu} A, \Gamma' \vdash x : A} \text{VAR} \quad \frac{\Gamma, x :_{\mu} A \vdash e : B}{\Gamma \vdash \lambda x. e : A^{\mu} \rightarrow B} \text{LAM} \quad \frac{\Gamma \vdash e : A^{\mu} \rightarrow B \quad \Gamma, \blacksquare_{\mu} \vdash v : A}{\Gamma \vdash e v : B} \text{APP} \\
\\
\frac{\Gamma, \blacksquare_{\mu} \vdash v : A}{\Gamma \vdash \text{box}_{\mu} v : \square_{\mu} A} \square\text{I} \quad \frac{\Gamma, \blacksquare_{\nu} \vdash v : \square_{\mu} A \quad \Gamma, x :_{\nu \circ \mu} A \vdash e : B}{\Gamma \vdash \text{let}_{\nu} \text{box}_{\mu} x = v \text{ in } e : B} \square\text{E} \\
\\
\frac{\Gamma \vdash e_1 : A \quad \Gamma, x :_{\text{id}} A \vdash e_2 : B}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : B} \text{LET} \quad \frac{}{\Gamma \vdash () : \mathbb{1}} \text{UNIT} \quad \frac{\Gamma \vdash v_1 : A \quad \Gamma \vdash v_2 : B}{\Gamma \vdash (v_1, v_2) : A \times B} \text{PAIR} \\
\\
\frac{\Gamma, \blacksquare_{\mu} \vdash v : A \times B \quad \Gamma, x :_{\mu} A, y :_{\mu} B \vdash e : C}{\Gamma \vdash \text{let}_{\mu}(x, y) = v \text{ in } e : C} \text{SPLIT} \quad \frac{\Gamma \vdash v : A}{\Gamma \vdash \text{inl } v : A + B} \text{INL} \\
\\
\frac{\Gamma \vdash v : B}{\Gamma \vdash \text{inr } v : A + B} \text{INR} \quad \frac{\Gamma, \blacksquare_{\mu} \vdash v : A + B \quad \Gamma, x :_{\mu} A \vdash e_1 : C \quad \Gamma, y :_{\mu} B \vdash e_2 : C}{\Gamma \vdash \text{case}_{\mu} v \{x.e_1; y.e_2\} : C} \text{CASE}
\end{array}$$

Call-by-value semantics.

$\boxed{e \rightsquigarrow e'}$

$$\begin{aligned}
& (\lambda x. e) v \rightsquigarrow e[v/x] \\
& \text{let } x = v \text{ in } e \rightsquigarrow e[v/x] \\
& \text{let}_{\nu} \text{box}_{\mu} x = \text{box}_{\mu} v \text{ in } e \rightsquigarrow e[v/x] \\
& \text{let}_{\mu}(x, y) = (v_1, v_2) \text{ in } e \rightsquigarrow e[v_1/x, v_2/y] \\
& \text{case}_{\mu}(\text{inl } v) \{x.e_1; y.e_2\} \rightsquigarrow e_1[v/x] \\
& \text{case}_{\mu}(\text{inr } v) \{x.e_1; y.e_2\} \rightsquigarrow e_2[v/y]
\end{aligned}$$

Evaluation contexts.

$$E ::= [\cdot] \mid E v \mid \text{let } x = E \text{ in } e$$

$$\frac{e \rightsquigarrow e'}{E[e] \rightsquigarrow E[e']} \text{CTX}$$

Parameterisation. Multi-modal type theory is usually parameterised by a two-category of modes, modalities and 2-cells. In the calculus above, we assume that we have a single mode and for each two modalities μ_1 and μ_2 , there is at most one 2-cell $\mu_1 \Rightarrow \mu_2$ between them. In that case, we can describe a mode theory as a monoid of modalities (with identity id and composition \circ) together with a preorder on modalities (with $\mu \Rightarrow \nu$ iff $\mu \leq \nu$).

To obtain the instance of the Fitch-style IS4 calculus, we take the two-element monoid $\{\text{id}, \blacksquare\}$ with $\blacksquare \circ \blacksquare = \blacksquare$ and $\blacksquare \leq \text{id}$.

Derivations. We derive terms for the following types:

$$\begin{aligned}
T &= \lambda x. \text{let}_{\text{id}} \text{box}_{\blacksquare} y = x \text{ in } y : \square_{\blacksquare} A^{\text{id}} \rightarrow A \\
4 &= \lambda x. \text{let}_{\text{id}} \text{box}_{\blacksquare} y = x \text{ in } \text{box}_{\blacksquare} (\text{box}_{\blacksquare} y) : \square_{\blacksquare} A^{\text{id}} \rightarrow \square_{\blacksquare} \square_{\blacksquare} A \\
N &= \text{box}_{\mu} () : \square_{\mu} \mathbb{1} \\
X &= \lambda p. \text{let}_{\text{id}}(x, y) = p \text{ in } \text{let}_{\text{id}} \text{box}_{\mu} f = x \text{ in } \text{let}_{\text{id}} \text{box}_{\mu} g = y \text{ in} \\
&\quad \text{box}_{\mu} (\lambda a. \text{let } b = f a \text{ in } g b) \\
&\quad : \square_{\mu}(A^{\text{id}} \rightarrow B) \times \square_{\mu}(B^{\text{id}} \rightarrow C)^{\text{id}} \rightarrow \square_{\mu}(A^{\text{id}} \rightarrow C) \\
M^{\times} &= \lambda p. \text{let}_{\text{id}}(x, y) = p \text{ in } \text{let}_{\text{id}} \text{box}_{\mu} a = x \text{ in} \\
&\quad \text{let}_{\text{id}} \text{box}_{\mu} b = y \text{ in } \text{box}_{\mu} (a, b) \\
&\quad : \square_{\mu} A \times \square_{\mu} B^{\text{id}} \rightarrow \square_{\mu}(A \times B) \\
M^{+} &= \lambda s. \text{case}_{\text{id}} s \{x. \text{let}_{\text{id}} \text{box}_{\mu} a = x \text{ in } \text{box}_{\mu} (\text{inl } a); \\
&\quad y. \text{let}_{\text{id}} \text{box}_{\mu} b = y \text{ in } \text{box}_{\mu} (\text{inr } b)\} \\
&\quad : \square_{\mu} A + \square_{\mu} B^{\text{id}} \rightarrow \square_{\mu}(A + B)
\end{aligned}$$

A.3 Graded Call-by-Push-Value

Syntax.

$$\begin{aligned}
A &::= \underline{UB} \mid \square^q A \mid A \times B \mid A + B \mid \mathbb{1} \\
\underline{B} &::= F^q A \mid {}^q A \rightarrow \underline{B} \\
v &::= x \mid \text{thunk } c \mid \text{box}_q v \mid (v_1, v_2) \mid () \mid \text{inl } v \mid \text{inr } v \\
c &::= \text{force } v \mid \lambda x. c \mid c v \mid \text{return}_q v \\
&\quad \mid \text{let } x \leftarrow c_1 \text{ in } c_2 \mid \text{let}_{q_1} \text{box}_{q_2} x = v \text{ in } c \\
&\quad \mid \text{let}_q(x, y) = v \text{ in } c \mid \text{case}_q v \{x.c_1; y.c_2\}
\end{aligned}$$

Value typing rules.

$$\boxed{\Gamma \vdash v : A}$$

$$\begin{array}{c}
\frac{}{0 \cdot \Gamma, x : {}^1 A, 0 \cdot \Gamma' \vdash x : A} \text{VAR} \qquad \frac{q \leq q' \quad \Gamma, x : {}^q A, \Gamma' \vdash v : B}{\Gamma, x : {}^{q'} A, \Gamma' \vdash v : B} \text{SUB} \\
\\
\frac{\Gamma \vdash c : \underline{B}}{\Gamma \vdash \text{thunk } c : \underline{UB}} \text{THUNK} \qquad \frac{}{\Gamma \vdash () : \mathbb{1}} \text{UNIT} \qquad \frac{\Gamma_1 \vdash v_1 : A \quad \Gamma_2 \vdash v_2 : B}{\Gamma_1 + \Gamma_2 \vdash (v_1, v_2) : A \times B} \text{PAIR} \\
\\
\frac{\Gamma \vdash v : A}{\Gamma \vdash \text{inl } v : A + B} \text{INL} \qquad \frac{\Gamma \vdash v : B}{\Gamma \vdash \text{inr } v : A + B} \text{INR} \qquad \frac{\Gamma \vdash v : A}{q \cdot \Gamma \vdash \text{box}_q v : \square^q A} \square\text{I}
\end{array}$$

Computation typing rules.

$$\boxed{\Gamma \vdash c : \underline{B}}$$

$$\begin{array}{c}
\frac{\Gamma \vdash v : U\underline{B}}{\Gamma \vdash \text{force } v : \underline{B}} \text{ FORCE} \quad \frac{\Gamma, x :^q A \vdash c : \underline{B}}{\Gamma \vdash \lambda x. c : ^q A \rightarrow \underline{B}} \text{ LAM} \quad \frac{\Gamma_1 \vdash c : ^q A \rightarrow \underline{B} \quad \Gamma_2 \vdash v : A}{\Gamma_1 + q \cdot \Gamma_2 \vdash c v : \underline{B}} \text{ APP} \\
\\
\frac{\Gamma \vdash v : A}{q \cdot \Gamma \vdash \text{return}_q v : F^q A} \text{ RETURN} \quad \frac{\Gamma_1 \vdash c_1 : F^q A \quad \Gamma_2, x :^q A \vdash c_2 : \underline{B}}{\Gamma_1 + \Gamma_2 \vdash \text{let } x \leftarrow c_1 \text{ in } c_2 : \underline{B}} \text{ BIND} \\
\\
\frac{\Gamma_1 \vdash v : \square^{q_2} A \quad \Gamma_2, x :^{q_1 \cdot q_2} A \vdash c : \underline{B}}{q_1 \cdot \Gamma_1 + \Gamma_2 \vdash \text{let}_{q_1} \text{box}_{q_2} x = v \text{ in } c : \underline{B}} \square E \quad \frac{\Gamma_1 \vdash v : A_1 \times A_2 \quad \Gamma_2, x :^q A_1, y :^q A_2 \vdash c : \underline{B}}{q \cdot \Gamma_1 + \Gamma_2 \vdash \text{let}_q(x, y) = v \text{ in } c : \underline{B}} \text{ SPLIT} \\
\\
\frac{\Gamma_1 \vdash v : A + B \quad \Gamma_2, x :^q A \vdash c_1 : \underline{C} \quad \Gamma_2, y :^q B \vdash c_2 : \underline{C}}{q \cdot \Gamma_1 + \Gamma_2 \vdash \text{case}_q v \{x.c_1; y.c_2\} : \underline{C}} \text{ CASE}
\end{array}$$

Semantics.

$$\boxed{c \rightsquigarrow c'}$$

$$\begin{array}{l}
\text{force (thunk } c) \rightsquigarrow c \\
(\lambda x. c) v \rightsquigarrow c[v/x] \\
\text{let } x \leftarrow \text{return}_q v \text{ in } c \rightsquigarrow c[v/x] \\
\text{let}_{q_1} \text{box}_{q_2} x = \text{box}_{q_2} v \text{ in } c \rightsquigarrow c[v/x] \\
\text{let}_q(x, y) = (v_1, v_2) \text{ in } c \rightsquigarrow c[v_1/x, v_2/y] \\
\text{case}_q(\text{inl } v) \{x.c_1; y.c_2\} \rightsquigarrow c_1[v/x] \\
\text{case}_q(\text{inr } v) \{x.c_1; y.c_2\} \rightsquigarrow c_2[v/y]
\end{array}$$

Evaluation contexts.

$$E ::= [\cdot] \mid E v \mid \text{let } x \leftarrow E \text{ in } c$$

$$\frac{c \rightsquigarrow c'}{E[c] \rightsquigarrow E[c']} \text{ CTX}$$

Parameterisation. Graded modal type theory is usually parameterised by a pre-ordered semiring $(S, 0, 1, +, \cdot, \leq)$ of grades. Aside from the usual semiring axioms, we require that $+$ and \cdot are monotone with respect to \leq .

To obtain the instance of the Fitch-style IS4 calculus, we take the three-element semiring $\{0, 1, \blacksquare\}$ with $0 \leq 1 \leq \blacksquare$, $+$ as maximum, and $\blacksquare \cdot \blacksquare = \blacksquare$.

Derivations. We derive terms for the following types:

$$\begin{aligned}
T &= \lambda x. \text{let}_1 \text{box}_\blacksquare y = x \text{ in return}_1 y : {}^1(\square^\blacksquare A) \rightarrow F^1 A \\
4 &= \lambda x. \text{let}_1 \text{box}_\blacksquare y = x \text{ in return}_1 (\text{box}_\blacksquare (\text{box}_\blacksquare y)) : {}^1(\square^\blacksquare A) \rightarrow F^1(\square^\blacksquare \square^\blacksquare A) \\
N &= \text{box}_q () : \square^q \mathbb{1} \\
X &= \lambda p. \text{let}_1(x, y) = p \text{ in let}_1 \text{box}_q f = x \text{ in let}_1 \text{box}_q g = y \text{ in} \\
&\quad \text{return}_1 (\text{box}_q (\text{thunk } (\lambda a. \text{let } b \leftarrow \text{force } f \text{ a in force } g b))) \\
&\quad : {}^1(\square^q U({}^1 A \rightarrow F^1 B) \times \square^q U({}^1 B \rightarrow F^1 C)) \\
&\quad \rightarrow F^1(\square^q U({}^1 A \rightarrow F^1 C)) \\
M^\times &= \lambda p. \text{let}_1(x, y) = p \text{ in let}_1 \text{box}_q a = x \text{ in} \\
&\quad \text{let}_1 \text{box}_q b = y \text{ in return}_1 (\text{box}_q (a, b)) \\
&\quad : {}^1(\square^q A \times \square^q B) \rightarrow F^1(\square^q(A \times B)) \\
M^+ &= \lambda s. \text{case}_1 s \{x. \text{let}_1 \text{box}_q a = x \text{ in return}_1 (\text{box}_q (\text{inl } a)); \\
&\quad y. \text{let}_1 \text{box}_q b = y \text{ in return}_1 (\text{box}_q (\text{inr } b))\} \\
&\quad : {}^1(\square^q A + \square^q B) \rightarrow F^1(\square^q(A + B))
\end{aligned}$$

A.4 Adjoint Natural Deduction

Syntax.

$$\begin{aligned}
A_m, B_m &::= A_m \multimap B_m \mid \uparrow_k^n A_k \quad (m \geq k) \\
&\mid A_m \otimes B_m \mid \mathbf{1}_m \mid A_m \oplus B_m \mid \downarrow_m^n A_n \quad (n \geq m) \\
e &::= s \mid \lambda x. e \mid \mathbf{susp } e \\
&\mid (e_1, e_2) \mid () \mid \text{inl } e \mid \text{inr } e \mid \mathbf{down } e \\
&\mid \mathbf{match } s ((x_1, x_2) \Rightarrow e) \\
&\mid \mathbf{match } s (() \Rightarrow e) \\
&\mid \mathbf{match } s (\text{inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \\
&\mid \mathbf{match } s (\mathbf{down } x \Rightarrow e) \\
s &::= x \mid (e : A_m) \mid s e \mid \mathbf{force } s \\
\Delta &::= \Delta_W \mid \Delta, x : A_m \mid \Delta ; \Delta
\end{aligned}$$

Typing rules.

$$\boxed{\Delta \vdash e \Leftarrow A_m}$$

$$\boxed{\Delta \vdash s \Rightarrow A_m}$$

$$\begin{array}{c}
\frac{\Delta \vdash s \Rightarrow A_m}{\Delta \vdash s \Leftarrow A_m} \Rightarrow / \Leftarrow \quad \frac{\Delta \vdash e \Leftarrow A_m}{\Delta \vdash (e : A_m) \Rightarrow A_m} \Leftarrow / \Rightarrow \quad \frac{m \geq n}{\Delta_W ; x : A_m \vdash x \Rightarrow A_n} \text{HYP} \\
\\
\frac{\Delta, x : A_m \vdash e \Leftarrow B_m}{\Delta \vdash \lambda x. e \Leftarrow A_m \multimap B_m} \multimap \text{I} \quad \frac{\Delta \vdash s \Rightarrow A_m \multimap B_m \quad \Delta' \vdash e \Leftarrow A_m}{\Delta ; \Delta' \vdash s e \Rightarrow B_m} \multimap \text{E} \\
\\
\frac{\Delta \vdash e \Leftarrow A_k}{\Delta \vdash \text{susp } e \Leftarrow \uparrow_k^m A_k} \uparrow \text{I} \quad \frac{\Delta' \geq m \quad \Delta' \vdash s \Rightarrow \uparrow_k^m A_k}{\Delta_W ; \Delta' \vdash \text{force } s \Rightarrow A_k} \uparrow \text{E} \\
\\
\frac{\Delta \vdash e_1 \Leftarrow A_m \quad \Delta' \vdash e_2 \Leftarrow B_m}{\Delta ; \Delta' \vdash (e_1, e_2) \Leftarrow A_m \otimes B_m} \otimes \text{I} \quad \frac{\Delta \vdash s \Rightarrow A_m \otimes B_m \quad \Delta \geq m \geq r \quad \Delta', x_1 : A_m, x_2 : B_m \vdash e' \Leftarrow C_r}{\Delta ; \Delta' \vdash \text{match } s ((x_1, x_2) \Rightarrow e') \Leftarrow C_r} \otimes \text{E} \\
\\
\frac{}{\Delta_W \vdash () \Leftarrow \mathbf{1}_m} \mathbf{1} \text{I} \quad \frac{\Delta \vdash s \Rightarrow \mathbf{1}_m \quad \Delta \geq m \geq r \quad \Delta' \vdash e' \Leftarrow C_r}{\Delta ; \Delta' \vdash \text{match } s (() \Rightarrow e') \Leftarrow C_r} \mathbf{1} \text{E} \\
\\
\frac{\Delta \vdash e \Leftarrow A_m}{\Delta \vdash \text{inl } e \Leftarrow A_m \oplus B_m} \oplus \text{I}_1 \quad \frac{\Delta \vdash e \Leftarrow B_m}{\Delta \vdash \text{inr } e \Leftarrow A_m \oplus B_m} \oplus \text{I}_2 \\
\\
\frac{\Delta \vdash s \Rightarrow A_m \oplus B_m \quad \Delta \geq m \geq r \quad \Delta', x : A_m \vdash e_1 \Leftarrow C_r \quad \Delta', y : B_m \vdash e_2 \Leftarrow C_r}{\Delta ; \Delta' \vdash \text{match } s (\text{inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \Leftarrow C_r} \oplus \text{E} \quad \frac{\Delta' \geq n \quad \Delta' \vdash e \Leftarrow A_n}{\Delta_W ; \Delta' \vdash \text{down } e \Leftarrow \downarrow_m^n A_n} \downarrow \text{I} \\
\\
\frac{\Delta \vdash s \Rightarrow \downarrow_m^n A_n \quad \Delta \geq m \geq r \quad \Delta', x : A_n \vdash e' \Leftarrow C_r}{\Delta ; \Delta' \vdash \text{match } s (\text{down } x \Rightarrow e') \Leftarrow C_r} \downarrow \text{E}
\end{array}$$

Call-by-value semantics.

$$e \rightsquigarrow e'$$

$$v ::= \lambda x. e \mid \text{susp } e \mid (v_1, v_2) \mid () \mid \text{inl } v \mid \text{inr } v \mid \text{down } v$$

$$\begin{array}{l}
((\lambda x. e) : A_m \multimap B_m) v \rightsquigarrow e[v/x] \\
\text{force } ((\text{susp } e) : \uparrow_k^m A_k) \rightsquigarrow e \\
\text{match } ((v_1, v_2) : A_m \otimes B_m) ((x_1, x_2) \Rightarrow e') \rightsquigarrow e'[v_1/x_1, v_2/x_2] \\
\text{match } (()) : \mathbf{1}_m \quad (() \Rightarrow e') \rightsquigarrow e' \\
\text{match } ((\text{inl } v) : A_m \oplus B_m) (\text{inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \rightsquigarrow e_1[v/x] \\
\text{match } ((\text{inr } v) : A_m \oplus B_m) (\text{inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \rightsquigarrow e_2[v/y] \\
\text{match } ((\text{down } v) : \downarrow_m^n A_n) (\text{down } x \Rightarrow e') \rightsquigarrow e'[v/x]
\end{array}$$

Evaluation contexts.

$$\begin{array}{l}
E ::= [\cdot] \mid (E : A_m) \mid E e \mid v E \mid \mathbf{force} E \\
\mid (E, e) \mid (v, E) \mid \mathbf{inl} E \mid \mathbf{inr} E \mid \mathbf{down} E \\
\mid \mathbf{match} E ((x_1, x_2) \Rightarrow e) \\
\mid \mathbf{match} E (() \Rightarrow e) \\
\mid \mathbf{match} E (\mathbf{inl} x \Rightarrow e_1 \mid \mathbf{inr} y \Rightarrow e_2) \\
\mid \mathbf{match} E (\mathbf{down} x \Rightarrow e)
\end{array}
\qquad
\frac{e \rightsquigarrow e'}{E[e] \rightsquigarrow E[e']} \text{CTX}$$

Parameterisation. Adjoint natural deduction is parameterised by a set of preordered modes (M, \leq) . The calculus maintains the invariant that if $\Delta \vdash e : A_m$ then $\Delta \geq m$. Each mode m may or may not allow for contraction and weakening, but if $m \leq n$ and m allows for either, then so must n . We write Δ_W to indicate that the modes of all variables in the context allow for weakening and $\Delta ; \Delta'$ to indicate that the modes of the variables in both contexts allow for contraction.

To obtain the instance of the Fitch-style IS4 calculus, we take the set of modes $\{1, \blacksquare\}$ with $1 \leq \blacksquare$ and both allowing for contraction and weakening.

Derivations. We derive terms for the following types:

$$\begin{array}{l}
T = \lambda x. \mathbf{match} x (\mathbf{down} y \Rightarrow y) : \downarrow_1^\blacksquare A_\blacksquare \multimap A_1 \\
4 = \lambda x. \mathbf{match} x (\mathbf{down} y \Rightarrow \mathbf{down} (\mathbf{down} y)) : \downarrow_1^\blacksquare A_\blacksquare \multimap \downarrow_1^\blacksquare (\downarrow_1^\blacksquare A_\blacksquare) \\
N = \mathbf{down} () : \downarrow_1^\blacksquare \mathbf{1}_\blacksquare \\
X = \lambda p. \mathbf{match} p ((f, g) \Rightarrow \\
\quad \mathbf{match} f (\mathbf{down} f' \Rightarrow \mathbf{match} g (\mathbf{down} g' \Rightarrow \\
\quad \mathbf{down} (\mathbf{susp} (\lambda a. (\mathbf{force} g') ((\mathbf{force} f') a)))))) \\
: \downarrow_1^\blacksquare (\uparrow_1^\blacksquare (A_1 \multimap B_1)) \otimes \downarrow_1^\blacksquare (\uparrow_1^\blacksquare (B_1 \multimap C_1)) \multimap \downarrow_1^\blacksquare (\uparrow_1^\blacksquare (A_1 \multimap C_1)) \\
M^\otimes = \lambda p. \mathbf{match} p ((x, y) \Rightarrow \\
\quad \mathbf{match} x (\mathbf{down} a \Rightarrow \mathbf{match} y (\mathbf{down} b \Rightarrow \\
\quad \mathbf{down} (a, b)))) \\
: \downarrow_1^\blacksquare A_\blacksquare \otimes \downarrow_1^\blacksquare B_\blacksquare \multimap \downarrow_1^\blacksquare (A_\blacksquare \otimes B_\blacksquare) \\
M^\oplus = \lambda s. \mathbf{match} s (\mathbf{inl} x \Rightarrow \mathbf{match} x (\mathbf{down} a \Rightarrow \mathbf{down} (\mathbf{inl} a)) \\
\quad \mid \mathbf{inr} y \Rightarrow \mathbf{match} y (\mathbf{down} b \Rightarrow \mathbf{down} (\mathbf{inr} b))) \\
: \downarrow_1^\blacksquare A_\blacksquare \oplus \downarrow_1^\blacksquare B_\blacksquare \multimap \downarrow_1^\blacksquare (A_\blacksquare \oplus B_\blacksquare)
\end{array}$$

A.5 Mode Qualifiers

Syntax.

$$\begin{array}{l}
A, B ::= A @ \mu_1 \rightarrow B @ \mu_2 \mid \square^v A \mid A \times B \mid A + B \mid \mathbb{1} \\
e ::= x \mid \lambda x. e \mid e_1 e_2 \mid \mathbf{box}_v e \mid \mathbf{unbox}_v e \mid (e_1, e_2) \mid () \mid \mathbf{inl} e \mid \mathbf{inr} e \\
\quad \mid \mathbf{let} x = e_1 \mathbf{in} e_2 \mid \mathbf{let}_\mu(x, y) = e \mathbf{in} e \mid \mathbf{case}_\mu e \{x.e_1; y.e_2\} \\
\Gamma ::= \cdot \mid \Gamma, x : A @ \mu, \blacksquare_\mu
\end{array}$$

We define $\text{locks}(\Gamma)$ as the composition of all locks in Γ :

$$\begin{aligned}\text{locks}(\cdot) &= \text{id} \\ \text{locks}(\Gamma, x :_{\mu} A) &= \text{locks}(\Gamma) \\ \text{locks}(\Gamma, \mathbf{\mu}_{\mu}) &= \mu \circ \text{locks}(\Gamma)\end{aligned}$$

Typing rules.

$$\boxed{\Gamma \vdash e : A @ \mu}$$

$$\begin{array}{c} \frac{\mu_1 \leq \text{locks}(\Gamma') \circ \mu_2}{\Gamma, x : A @ \mu_1, \Gamma' \vdash x : A @ \mu_2} \text{VAR} \qquad \frac{\Gamma, \mathbf{\mu}_{\mu}, x : A @ \mu_1 \vdash e : B @ \mu_2}{\Gamma \vdash \lambda x. e : (A @ \mu_1 \rightarrow B @ \mu_2) @ \mu} \text{LAM} \\ \\ \frac{\Gamma \vdash e_1 : (A @ \mu_1 \rightarrow B @ \mu_2) @ _}{\Gamma \vdash e_2 : A @ \mu_1} \text{APP} \qquad \frac{\Gamma \vdash e : A @ v \circ \mu}{\Gamma \vdash \text{box}_v e : \square^v A @ \mu} \square\text{I} \\ \\ \frac{\Gamma \vdash e : \square^v A @ \mu}{\Gamma \vdash \text{unbox}_v e : A @ v \circ \mu} \square\text{E} \qquad \frac{\Gamma \vdash e_1 : A @ \mu_1 \quad \Gamma, x : A @ \mu_1 \vdash e_2 : B @ \mu_2}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 : B @ \mu_2} \text{LET} \\ \\ \frac{}{\Gamma \vdash () : \mathbf{1} @ \mu} \text{UNIT} \qquad \frac{\Gamma \vdash e_1 : A @ \mu \quad \Gamma \vdash e_2 : B @ \mu}{\Gamma \vdash (e_1, e_2) : A \times B @ \mu} \text{PAIR} \\ \\ \frac{\Gamma \vdash e_1 : A \times B @ \mu_1 \quad \Gamma, x : A @ \mu_1, y : B @ \mu_1 \vdash e_2 : C @ \mu_2}{\Gamma \vdash \text{let}_{\mu_1}(x, y) = e_1 \text{ in } e_2 : C @ \mu_2} \text{SPLIT} \qquad \frac{\Gamma \vdash e : A @ \mu}{\Gamma \vdash \text{inl } e : A + B @ \mu} \text{INL} \\ \\ \frac{\Gamma \vdash e : B @ \mu}{\Gamma \vdash \text{inr } e : A + B @ \mu} \text{INR} \qquad \frac{\Gamma \vdash e : A + B @ \mu_1 \quad \Gamma, x : A @ \mu_1 \vdash e_1 : C @ \mu_2 \quad \Gamma, y : B @ \mu_1 \vdash e_2 : C @ \mu_2}{\Gamma \vdash \text{case}_{\mu_1} e \{x.e_1; y.e_2\} : C @ \mu_2} \text{CASE}\end{array}$$

Call-by-value semantics.

$$\boxed{e \rightsquigarrow e'}$$

$$v ::= x \mid \lambda x. e \mid \text{box}_v v \mid (v_1, v_2) \mid () \mid \text{inl } v \mid \text{inr } v$$

$$\begin{aligned}(\lambda x. e) v &\rightsquigarrow e[v/x] \\ \text{let } x = v \text{ in } e &\rightsquigarrow e[v/x] \\ \text{unbox}_v (\text{box}_v v) &\rightsquigarrow v \\ \text{let}_{\mu}(x, y) = (v_1, v_2) \text{ in } e &\rightsquigarrow e[v_1/x, v_2/y] \\ \text{case}_{\mu} (\text{inl } v) \{x.e_1; y.e_2\} &\rightsquigarrow e_1[v/x] \\ \text{case}_{\mu} (\text{inr } v) \{x.e_1; y.e_2\} &\rightsquigarrow e_2[v/y]\end{aligned}$$

Evaluation contexts.

$$\begin{aligned}
E ::= [\cdot] \mid \text{box}_v E \mid (E, e) \mid (v, E) \mid \text{inl } E \mid \text{inr } E \\
\mid E e_2 \mid v_1 E \mid \text{unbox}_v E \\
\mid \text{let } x = E \text{ in } e \mid \text{let}_\mu(x, y) = E \text{ in } e \\
\mid \text{case}_\mu E \{x.e_1; y.e_2\}
\end{aligned}
\qquad
\frac{e \rightsquigarrow e'}{E[e] \rightsquigarrow E[e']} \text{CTX}$$

Parameterisation. Mode qualifier systems are usually parameterised by a set of axes \mathcal{A} . Each axis $((A, \leq) \in \mathcal{A})$ is a total order and contains a distinguished element 1_A . The distinguished element 1_A is either the minimum element of A (in which we call this axis *monadic*) or the maximum element of A (in which case we call this axis *comonadic*). Composition is given by maximum on monadic axes and by minimum on comonadic axes. A mode μ is a product of axes which is partially ordered by the pointwise extension of the orders on the axes.

To obtain the instance of the Fitch-style IS4 calculus, we take a single comonadic axis $\{1, \blacksquare\}$ and $\blacksquare \leq 1$. For this parameterisation, it is also possible to define the lock as a left-division operator:

$$\begin{aligned}
\cdot, \blacksquare_\mu &= \cdot \\
\Gamma, x :_{\mu_1} A, \blacksquare_{\mu_2} &= \Gamma, \blacksquare_{\mu_2}, x :_{\mu_1} A && (\text{if } \mu_1 \leq \mu_2) \\
\Gamma, x :_{\mu_1} A, \blacksquare_{\mu_2} &= \Gamma, \blacksquare_{\mu_2} && (\text{otherwise})
\end{aligned}$$

Derivations. We derive terms for the following types:

$$\begin{aligned}
T &= \lambda x. \text{unbox}_{\blacksquare} x : ((\square^{\blacksquare} A) @ 1 \rightarrow A @ 1) @ 1 \\
4 &= \lambda x. \text{box}_{\blacksquare} (\text{box}_{\blacksquare} (\text{unbox}_{\blacksquare} x)) : ((\square^{\blacksquare} A) @ 1 \rightarrow (\square^{\blacksquare} \square^{\blacksquare} A) @ 1) @ 1 \\
N &= \text{box}_v () : \square^v \mathbb{1} @ 1 \\
X &= \lambda p. \text{let}_1(f, g) = p \text{ in let } f' = \text{unbox}_v f \text{ in let } g' = \text{unbox}_v g \text{ in} \\
&\quad \text{box}_v (\lambda a. g' (f' a)) \\
&\quad : ((\square^v(A @ 1 \rightarrow B @ 1)) \times (\square^v(B @ 1 \rightarrow C @ 1))) @ 1 \\
&\quad \rightarrow (\square^v(A @ 1 \rightarrow C @ 1)) @ 1 @ 1 \\
M^\times &= \lambda p. \text{let}_1(x, y) = p \text{ in box}_v (\text{unbox}_v x, \text{unbox}_v y) \\
&\quad : ((\square^v A \times \square^v B) @ 1 \rightarrow \square^v(A \times B) @ 1) @ 1 \\
M^+ &= \lambda s. \text{case}_1 s \{x. \text{box}_v (\text{inl} (\text{unbox}_v x)); \\
&\quad \quad y. \text{box}_v (\text{inr} (\text{unbox}_v y))\} \\
&\quad : ((\square^v A + \square^v B) @ 1 \rightarrow \square^v(A + B) @ 1) @ 1
\end{aligned}$$

B Correspondences

We define translations between the types and terms of the calculi.

B.1 Converting Fitch-style IS4 to Let-style Unbox in Fine-grained CBV

Interpretation of types:

$$\begin{aligned}
\llbracket \mathbf{1} \rrbracket &= \mathbf{1} \\
\llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket \\
\llbracket A \times B \rrbracket &= \llbracket A \rrbracket \times \llbracket B \rrbracket \\
\llbracket A \rightarrow B \rrbracket &= \llbracket A \rrbracket^{\text{id}} \rightarrow \llbracket B \rrbracket \\
\llbracket \Box A \rrbracket &= \Box_{\blacksquare} \llbracket A \rrbracket
\end{aligned}$$

Contexts:

$$\begin{aligned}
\llbracket \cdot \rrbracket &= \cdot \\
\llbracket \Gamma, x : \Box A \rrbracket &= \llbracket \Gamma \rrbracket, x :_{\blacksquare} \llbracket A \rrbracket \\
\llbracket \Gamma, x : A \rrbracket &= \llbracket \Gamma \rrbracket, x :_{\text{id}} \llbracket A \rrbracket \\
\llbracket \Gamma, \blacksquare \rrbracket &= \llbracket \Gamma \rrbracket, \blacksquare_{\blacksquare}
\end{aligned}$$

Translation of values:

$$\begin{aligned}
\llbracket x \rrbracket &= \text{box } x \\
\llbracket () \rrbracket &= () \\
\llbracket \lambda x. e \rrbracket &= \lambda x'. \text{let}_{\mu} x = x' \text{ in } \llbracket e \rrbracket \\
\llbracket \text{box } v \rrbracket &= \text{box}_{\blacksquare} \llbracket v \rrbracket \\
\llbracket \text{unbox } x \rrbracket &= x \\
\llbracket \text{unbox (box } v) \rrbracket &= \llbracket v \rrbracket \\
\llbracket (v_1, v_2) \rrbracket &= (\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) \\
\llbracket \text{inl } v \rrbracket &= \text{inl } \llbracket v \rrbracket \\
\llbracket \text{inr } v \rrbracket &= \text{inr } \llbracket v \rrbracket
\end{aligned}$$

Translation of expressions:

$$\begin{aligned}
\llbracket e_1 e_2 \rrbracket &= \text{let } x = \llbracket e_1 \rrbracket \text{ in let } y = \llbracket e_2 \rrbracket \text{ in } x y \\
\llbracket \text{let } x = e_1 \text{ in } e_2 \rrbracket &= \text{let } x' = \llbracket e_1 \rrbracket \text{ in let}_{\mu} x = x' \text{ in } \llbracket e_2 \rrbracket \\
\llbracket \text{let } (x, y) = e_1 \text{ in } e_2 \rrbracket &= \text{let } z = \llbracket e_1 \rrbracket \text{ in let}_{\text{id}} (x', y') = z \text{ in let}_{\mu} x = x' \text{ in let}_{\mu} y = y' \text{ in } \llbracket e_2 \rrbracket \\
\llbracket \text{case } e \{ x.e_1; y.e_2 \} \rrbracket &= \text{let } z = \llbracket e \rrbracket \text{ in case}_{\text{id}} z \{ x'. \text{let}_{\mu} x = x' \text{ in } \llbracket e_1 \rrbracket; y'. \text{let}_{\mu} y = y' \text{ in } \llbracket e_2 \rrbracket \} \\
\llbracket \text{dist}^{\times} e \rrbracket &= \text{let } z = \llbracket e \rrbracket \text{ in let}_{\text{id}} \text{box}_{\blacksquare} w = z \text{ in} \\
&\quad \text{let}_{\blacksquare} (x, y) = w \text{ in } (\text{box}_{\blacksquare} x, \text{box}_{\blacksquare} y) \\
\llbracket \text{dist}^{+} e \rrbracket &= \text{let } z = \llbracket e \rrbracket \text{ in let}_{\text{id}} \text{box}_{\blacksquare} w = z \text{ in} \\
&\quad \text{case}_{\blacksquare} w \{ x. \text{inl } (\text{box}_{\blacksquare} x); y. \text{inr } (\text{box}_{\blacksquare} y) \}
\end{aligned}$$

CONJECTURE B.1 (TRANSLATION TO FGCBV). *If $\Gamma \vdash e : A$, then $\llbracket \Gamma \rrbracket \vdash \llbracket e \rrbracket : \llbracket A \rrbracket$ and if $e \rightsquigarrow^* v$, then $\llbracket e \rrbracket \rightsquigarrow^* \llbracket v \rrbracket$.*

B.2 Converting Let-style Unbox in Fine-grained CBV to Graded Call-by-Push-Value

Interpretation of types:

$$\begin{aligned}
\llbracket \mathbf{1} \rrbracket &= \mathbf{1} \\
\llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket \\
\llbracket A \times B \rrbracket &= \llbracket A \rrbracket \times \llbracket B \rrbracket \\
\llbracket \square_{\mu} A \rrbracket &= \square^{\mu} \llbracket A \rrbracket \\
\llbracket A^{\mu} \rightarrow B \rrbracket &= U(\mu \llbracket A \rrbracket \rightarrow F^1 \llbracket B \rrbracket)
\end{aligned}$$

Contexts:

$$\begin{aligned}
\llbracket \cdot \rrbracket_{\mu} &= \cdot \\
\llbracket \Gamma, x :_{\mu} A \rrbracket_1 &= \llbracket \Gamma \rrbracket_1, x :^{\mu} \llbracket A \rrbracket \\
\llbracket \Gamma, x :_1 A \rrbracket_{\blacksquare} &= \llbracket \Gamma \rrbracket_{\blacksquare}, x :^0 \llbracket A \rrbracket \\
\llbracket \Gamma, x :_{\blacksquare} A \rrbracket_{\blacksquare} &= \llbracket \Gamma \rrbracket_{\blacksquare}, x :^1 \llbracket A \rrbracket \\
\llbracket \Gamma, \blacksquare_{\mu} \rrbracket_{\nu} &= \llbracket \Gamma \rrbracket_{\mu \circ \nu}
\end{aligned}$$

Translation of values:

$$\begin{aligned}
\llbracket x \rrbracket &= x \\
\llbracket () \rrbracket &= () \\
\llbracket \lambda x. e \rrbracket &= \text{thunk}(\lambda x. \llbracket e \rrbracket) \\
\llbracket \text{box}_{\mu} v \rrbracket &= \text{box}_{\mu} \llbracket v \rrbracket \\
\llbracket (v_1, v_2) \rrbracket &= (\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) \\
\llbracket \text{inl } v \rrbracket &= \text{inl } \llbracket v \rrbracket \\
\llbracket \text{inr } v \rrbracket &= \text{inr } \llbracket v \rrbracket
\end{aligned}$$

Translation of expressions:

$$\begin{aligned}
\llbracket v \rrbracket &= \text{return}_1 \llbracket v \rrbracket \\
\llbracket e v \rrbracket &= \text{let } f \leftarrow \llbracket e \rrbracket \text{ in (force } f) \llbracket v \rrbracket \\
\llbracket \text{let } x = e_1 \text{ in } e_2 \rrbracket &= \text{let } x \leftarrow \llbracket e_1 \rrbracket \text{ in } \llbracket e_2 \rrbracket \\
\llbracket \text{let}_{\nu} \text{box}_{\mu} x = v \text{ in } e \rrbracket &= \text{let}_{\nu} \text{box}_{\mu} x = \llbracket v \rrbracket \text{ in } \llbracket e \rrbracket \\
\llbracket \text{let}_{\mu}(x, y) = v \text{ in } e \rrbracket &= \text{let}_{\mu}(x, y) = \llbracket v \rrbracket \text{ in } \llbracket e \rrbracket \\
\llbracket \text{case}_{\mu} v \{x.e_1; y.e_2\} \rrbracket &= \text{case}_{\mu} \llbracket v \rrbracket \{x. \llbracket e_1 \rrbracket; y. \llbracket e_2 \rrbracket\}
\end{aligned}$$

CONJECTURE B.2 (TRANSLATION TO GRADED CBPV). *If $\Gamma \vdash v : A$ (value judgment), then $\llbracket \Gamma \rrbracket_1 \vdash \llbracket v \rrbracket : \llbracket A \rrbracket$. If $\Gamma \vdash e : A$ (expression judgment), then $\llbracket \Gamma \rrbracket_1 \vdash \llbracket e \rrbracket : F^1 \llbracket A \rrbracket$. If $e \rightsquigarrow^* v$, then $\llbracket e \rrbracket \rightsquigarrow^* \llbracket v \rrbracket$.*

B.3 Converting Graded Call-by-Push-Value to Adjoint Natural Deduction

Interpretation of value types:

$$\begin{aligned}
\llbracket \mathbf{1} \rrbracket_m &= \mathbf{1}_m \\
\llbracket A + B \rrbracket_m &= \llbracket A \rrbracket_m \oplus \llbracket B \rrbracket_m \\
\llbracket A \times B \rrbracket_m &= \llbracket A \rrbracket_m \otimes \llbracket B \rrbracket_m \\
\llbracket \square^q A \rrbracket_m &= \downarrow_m^{q \cdot m} \llbracket A \rrbracket_{q \cdot m} \\
\llbracket U \underline{B} \rrbracket_m &= \uparrow_1^m \llbracket \underline{B} \rrbracket
\end{aligned}$$

Interpretation of computation types:

$$\begin{aligned}
\llbracket F^q A \rrbracket &= \downarrow_1^q \llbracket A \rrbracket_q \\
\llbracket {}^q A \rightarrow \underline{B} \rrbracket &= \llbracket A \rrbracket_q \multimap \llbracket \underline{B} \rrbracket
\end{aligned}$$

Contexts:

$$\begin{aligned}
\llbracket \cdot \rrbracket &= \cdot \\
\llbracket \Gamma, x :^0 A \rrbracket &= \llbracket \Gamma \rrbracket \\
\llbracket \Gamma, x :^q A \rrbracket &= \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket_q
\end{aligned}$$

Translation of values:

$$\begin{aligned}
\llbracket x \rrbracket &= x \\
\llbracket () \rrbracket &= () \\
\llbracket \text{think } c \rrbracket &= \mathbf{susp} \llbracket c \rrbracket \\
\llbracket \text{box}_q v \rrbracket &= \mathbf{down} \llbracket v \rrbracket \\
\llbracket (v_1, v_2) \rrbracket &= (\llbracket v_1 \rrbracket, \llbracket v_2 \rrbracket) \\
\llbracket \text{inl } v \rrbracket &= \mathbf{inl} \llbracket v \rrbracket \\
\llbracket \text{inr } v \rrbracket &= \mathbf{inr} \llbracket v \rrbracket
\end{aligned}$$

Translation of computations:

$$\begin{aligned}
\llbracket \text{force } v \rrbracket &= \mathbf{force} \llbracket v \rrbracket \\
\llbracket \lambda x. c \rrbracket &= \lambda x. \llbracket c \rrbracket \\
\llbracket c v \rrbracket &= \llbracket c \rrbracket \llbracket v \rrbracket \\
\llbracket \text{return}_q v \rrbracket &= \mathbf{down} \llbracket v \rrbracket \\
\llbracket \text{let } x \leftarrow c_1 \text{ in } c_2 \rrbracket &= \mathbf{match} \llbracket c_1 \rrbracket (\mathbf{down} x \Rightarrow \llbracket c_2 \rrbracket) \\
\llbracket \text{let}_{q_1} \text{box}_{q_2} x = v \text{ in } c \rrbracket &= \mathbf{match} \llbracket v \rrbracket (\mathbf{down} x \Rightarrow \llbracket c \rrbracket) \\
\llbracket \text{let}_q (x, y) = v \text{ in } c \rrbracket &= \mathbf{match} \llbracket v \rrbracket ((x, y) \Rightarrow \llbracket c \rrbracket) \\
\llbracket \text{case}_q v \{x.c_1; y.c_2\} \rrbracket &= \mathbf{match} \llbracket v \rrbracket (\mathbf{inl } x \Rightarrow \llbracket c_1 \rrbracket \mid \mathbf{inr } y \Rightarrow \llbracket c_2 \rrbracket)
\end{aligned}$$

CONJECTURE B.3 (TRANSLATION TO AND). *If $\Gamma \vdash v : A$, then $\llbracket \Gamma \rrbracket \vdash \llbracket v \rrbracket \Leftarrow \llbracket A \rrbracket$. If $\Gamma \vdash c : \underline{B}$, then $\llbracket \Gamma \rrbracket \vdash \llbracket c \rrbracket \Leftarrow \llbracket \underline{B} \rrbracket$. If $c \rightsquigarrow^* v$, then $\llbracket c \rrbracket \rightsquigarrow^* \llbracket v \rrbracket$.*

B.4 Converting Adjoint Natural Deduction to Mode Qualifiers

Interpretation of types:

$$\begin{aligned}
\llbracket \mathbf{1}_m \rrbracket &= \mathbb{1} \\
\llbracket A_m \oplus B_m \rrbracket &= \llbracket A_m \rrbracket + \llbracket B_m \rrbracket \\
\llbracket A_m \otimes B_m \rrbracket &= \llbracket A_m \rrbracket \times \llbracket B_m \rrbracket \\
\llbracket A_m \multimap B_m \rrbracket &= \llbracket A_m \rrbracket @ \llbracket m \rrbracket \rightarrow \llbracket B_m \rrbracket @ \llbracket m \rrbracket \\
\llbracket \downarrow_m^\blacksquare A_\blacksquare \rrbracket &= \square^\blacksquare \llbracket A_\blacksquare \rrbracket \\
\llbracket \downarrow_1^1 A_1 \rrbracket &= \llbracket A_1 \rrbracket \\
\llbracket \uparrow_k^m A_k \rrbracket &= \mathbb{1} @ 1 \rightarrow \llbracket A_k \rrbracket @ \llbracket k \rrbracket
\end{aligned}$$

Contexts:

$$\begin{aligned}
\llbracket \cdot \rrbracket &= \cdot \\
\llbracket \Gamma, x : A_m \rrbracket &= \llbracket \Gamma \rrbracket, x : \llbracket A \rrbracket @ \llbracket m \rrbracket
\end{aligned}$$

Translation of values:

$$\begin{aligned}
\llbracket () \rrbracket &= () \\
\llbracket x \rrbracket &= x \\
\llbracket \text{inl } e \rrbracket &= \text{inl } \llbracket e \rrbracket \\
\llbracket \text{inr } e \rrbracket &= \text{inr } \llbracket e \rrbracket \\
\llbracket (e_1, e_2) \rrbracket &= (\llbracket e_1 \rrbracket, \llbracket e_2 \rrbracket) \\
\llbracket \lambda x. e \rrbracket &= \lambda x. \llbracket e \rrbracket \\
\llbracket \text{down}_\blacksquare e \rrbracket &= \text{box}_\blacksquare \llbracket e \rrbracket \\
\llbracket \text{down}_1 e \rrbracket &= \llbracket e \rrbracket \\
\llbracket \text{susp } e \rrbracket &= \lambda x. \llbracket e \rrbracket
\end{aligned}$$

Translation of expressions:

$$\begin{aligned}
\llbracket \text{match } s \text{ (inl } x \Rightarrow e_1 \mid \text{inr } y \Rightarrow e_2) \rrbracket &= \text{case } \llbracket s \rrbracket \{x. \llbracket e_1 \rrbracket; y. \llbracket e_2 \rrbracket\} \\
\llbracket \text{match } s \text{ } ((x, y) \Rightarrow e) \rrbracket &= \text{let } (x, y) = \llbracket s \rrbracket \text{ in } \llbracket e \rrbracket \\
\llbracket \text{match } s \text{ (down}_\blacksquare x \Rightarrow e) \rrbracket &= \text{let } x = \text{unbox}_\blacksquare \llbracket s \rrbracket \text{ in } \llbracket e \rrbracket \\
\llbracket \text{match } s \text{ (down}_1 x \Rightarrow e) \rrbracket &= \text{let } x = \llbracket s \rrbracket \text{ in } \llbracket e \rrbracket \\
\llbracket \text{match } s \text{ } () \Rightarrow e \rrbracket &= \text{let } x = \llbracket s \rrbracket \text{ in } \llbracket e \rrbracket \\
\llbracket \text{force } s \rrbracket &= \llbracket s \rrbracket () \\
\llbracket s e \rrbracket &= \llbracket s \rrbracket \llbracket e \rrbracket
\end{aligned}$$

CONJECTURE B.4 (TRANSLATION FROM AND). *If $\Delta \vdash e \Leftarrow A_m$, then $\llbracket \Delta \rrbracket \vdash \llbracket e \rrbracket : \llbracket A \rrbracket @ \llbracket m \rrbracket$ and if $e \rightsquigarrow^* v$, then $\llbracket e \rrbracket \rightsquigarrow^* \llbracket v \rrbracket$.*

B.5 Converting Mode Qualifiers to Fitch-style IS4

To keep our interpretation as generic as possible, we write $\square^1 A$ for A and $\square^\blacksquare A$ for $\square A$. Similarly, for terms, we write $\text{box}_1 e$ for e and $\text{unbox}_1 e$ for e .

Interpretation of types:

$$\begin{aligned}
\llbracket \mathbf{1} \rrbracket &= \mathbf{1} \\
\llbracket A + B \rrbracket &= \llbracket A \rrbracket + \llbracket B \rrbracket \\
\llbracket A \times B \rrbracket &= \llbracket A \rrbracket \times \llbracket B \rrbracket \\
\llbracket A @ \mu_1 \rightarrow B @ \mu_2 \rrbracket &= \square^{\mu_1} \llbracket A \rrbracket \rightarrow \square^{\mu_2} \llbracket B \rrbracket \\
\llbracket \square^\mu A \rrbracket &= \square^\mu \llbracket A \rrbracket
\end{aligned}$$

Contexts:

$$\begin{aligned}
\llbracket \cdot \rrbracket &= \cdot \\
\llbracket \Gamma, x : A @ \mu \rrbracket &= \llbracket \Gamma \rrbracket, x : \square^\mu \llbracket A \rrbracket \\
\llbracket \Gamma, \blacksquare_\bullet \rrbracket &= \llbracket \Gamma \rrbracket, \blacksquare_\bullet \\
\llbracket \Gamma, \blacksquare_1 \rrbracket &= \llbracket \Gamma \rrbracket
\end{aligned}$$

Translation of expressions:

$$\begin{aligned}
\llbracket x \rrbracket &= x \\
\llbracket () \rrbracket &= () \\
\llbracket \text{box}_\mu e \rrbracket &= \llbracket e \rrbracket \\
\llbracket \lambda x. e \rrbracket &= \text{box}(\lambda x'. \llbracket e \rrbracket [(\text{unbox } x')/x]) \\
\llbracket (e_1, e_2) \rrbracket &= \text{let } x = \llbracket e_1 \rrbracket \text{ in let } y = \llbracket e_2 \rrbracket \text{ in box}(\text{unbox } x, \text{unbox } y) \\
\llbracket \text{inl } e \rrbracket &= \text{let } x = \llbracket e \rrbracket \text{ in box inl } (\text{unbox } x) \\
\llbracket \text{inr } e \rrbracket &= \text{let } x = \llbracket e \rrbracket \text{ in box inr } (\text{unbox } x) \\
\llbracket e_1 e_2 \rrbracket &= \text{let } x = \llbracket e_1 \rrbracket \text{ in}(\text{unbox } x) \llbracket e_2 \rrbracket \\
\llbracket \text{let } x = e_1 \text{ in } e_2 \rrbracket &= \text{let } x' = \llbracket e_1 \rrbracket \text{ in} \llbracket e_2 \rrbracket [(\text{unbox } x')/x] \\
\llbracket \text{unbox}_\bullet e \rrbracket &= \text{let } x = \llbracket e \rrbracket \text{ in}(\text{unbox } x) \\
\llbracket \text{unbox}_1 e \rrbracket &= \llbracket e \rrbracket \\
\llbracket \text{let}_\bullet(x, y) = e_1 \text{ in } e_2 \rrbracket &= \text{let}(x', y') = (\text{dist}^\times \llbracket e_1 \rrbracket) \text{ in} \llbracket e_2 \rrbracket [(\text{unbox } x')/x, (\text{unbox } y')/y] \\
\llbracket \text{let}_1(x, y) = e_1 \text{ in } e_2 \rrbracket &= \text{let}(x', y') = \llbracket e_1 \rrbracket \text{ in} \llbracket e_2 \rrbracket [(\text{unbox } x')/x, (\text{unbox } y')/y] \\
\llbracket \text{case}_\bullet e \{x.e_1; y.e_2\} \rrbracket &= \text{case}(\text{dist}^+ \llbracket e \rrbracket) \{x'. \llbracket e_1 \rrbracket [(\text{unbox } x')/x]; y'. \llbracket e_2 \rrbracket [(\text{unbox } y')/y]\} \\
\llbracket \text{case}_1 e \{x.e_1; y.e_2\} \rrbracket &= \text{case} \llbracket e \rrbracket \{x'. \llbracket e_1 \rrbracket [(\text{unbox } x')/x]; y'. \llbracket e_2 \rrbracket [(\text{unbox } y')/y]\}
\end{aligned}$$

CONJECTURE B.5 (TRANSLATION TO IS4). *If $\Gamma \vdash e : A @ \mu$, then $\llbracket \Gamma \rrbracket \vdash \llbracket e \rrbracket : \square^\mu \llbracket A \rrbracket$ and if $e \rightsquigarrow^* v$, then $\llbracket e \rrbracket \rightsquigarrow^* \llbracket v \rrbracket$.*