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 identify the *value trick* as a unifying principle connecting several approaches from the literature.

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 [26] 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:

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

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 unbox e, the inner term e is evaluated until a suitable value box e is reached. In contrast, the term in the box is not evaluated directly and is returned by the elimination rule:

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

Such semantics have been used to track properties of computations such as mobility [20], co-effects [25], erasability [1, 3], usage in call-by-name [1, 6, 21], and staged computation [11]. However, as was already remarked by Murphy et al. [20], 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:

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

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 [12, 19], affinity [19], purity [7], acyclicity [16], and effects [28, 29].

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 $(\Box(A \to B) \to (\Box A \to \Box B))$. However, this axiom does not hold if the calculus contains primitives that can generate new resources out of thin air. For example, if $\Box A$ denotes non-linear values of a linear type A, then $\Box \mathbb{1}$ and $\Box(\mathbb{1} \to \text{File})$ are inhabited by unit and openFile, but $\Box\text{File}$ should not be inhabited, as files should be kept linear.

Authors' Contact Information: Anton Lorenzen, University of Edinburgh, United Kingdom, anton.lorenzen@ed.ac.uk; Wenhao Tang, University of Edinburgh, United Kingdom, wenhao.tang@ed.ac.uk; Sam Lindley, University of Edinburgh, United Kingdom, Sam.Lindley@ed.ac.uk.

The Continuation Trick. Wadler [31] solves this problem by using a continuation-passing style interface for primitives that generate new resources, like withFile : (File → File × X) → X. The interface ensures that no function generates a linear resource out of thin air; in other words, resources that may not go under the box have no introduction form, and are instead only accessible to a program if passed by its caller [7].

The Value Trick. In this work, we propose a different solution. We observe that resources like File are typically introduced by primitives like openFile (), which are expressions rather than values. This is not incidental, because generation of new resources is inherently a side-effecting operation. We can thus ensure that these resources are never boxed by restricting the box rule to values.

We are not the first to propose this idea: several authors have proposed to restrict the box rule to values in the past [10, 19], or used a version of this trick in their work [2, 5, 28–30]. However, we believe that the full consequences of this design have not been fully explored before. In this abstract, we show that the value trick applies in various settings and unifies several designs from the literature.

Fitch-style Unbox. To apply the value trick to the Fitch-style presentation of modal type systems, we have to restrict both the box and unbox rules to values:

$$\frac{🔒 \notin \Gamma'}{\Gamma, x : A, \Gamma' \vdash x : A'} \qquad \frac{\Gamma, 🔒 \vdash v : A}{\Gamma \vdash \mathrm{box}\ v : \Box A} \qquad \frac{\Gamma \vdash v : \Box A}{\Gamma, \Gamma' \vdash \mathrm{unbox}\ v : A}$$

Furthermore, both box v and unbox v are part of the syntax of values, which makes it possible to unbox a variable under a box. While unbox (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" [17].

Let-style Unbox in Fine-grained CBV. We can generalise the value trick to multimodal type theory (MTT) [14] in fine-grain call-by-value style [18]. Variants of this calculus were used by Ahman [2], Tang et al. [29] and Tang and Lindley [28]. 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 $\mathrm{locks}(\Gamma')$ for the composition of the modalities of the locks in Γ'. The relevant rules are:

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

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 incorporates a lock in the context, its term must be a value. However, we could allow expressions in the unbox rule if we removed the $🔒_{\nu}$ from the context in the premise and set ν to be the identity modality.

Graded Call-by-Push-Value. Torczon et al. [30] propose an elegant formulation of graded [1, 6] type theory in call-by-push-value [17] where the grades attach to values. Graded type theory is parameterised by a semiring of grades q, which can be composed using + and ·. 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:

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

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 must be a value. However, Torczon et al. [30] actually do not follow this recipe 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.

Modal Type 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 modal type qualifiers [19] (related to the earlier work of [12, 13, 22, 23]):

$$\frac{\mu_1 \leq \mathrm{locks}(\Gamma') \circ \mu_2}{\Gamma, x : A \mathbin{@} \mu_1, \Gamma' \vdash x : A \mathbin{@} \mu_2}\ \textsc{var} \qquad \frac{\Gamma \vdash v : A \mathbin{@} \mu \circ \nu}{\Gamma \vdash \mathrm{box}_\nu\ v : \Box^\nu A \mathbin{@} \mu}\ \textsc{box}$$

$$\frac{\Gamma \vdash e_1 : A \mathbin{@} \mu \qquad \Gamma \vdash e_2 : B \mathbin{@} \mu}{\Gamma \vdash (e_1, e_2) : A \times B \mathbin{@} \mu}\ \textsc{pair} \qquad \frac{\Gamma, \text{🔒}_\mu, x : A \mathbin{@} \mu_1 \vdash e : B \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (A \mathbin{@} \mu_1 \to B \mathbin{@} \mu_2) \mathbin{@} \mu}\ \textsc{lam}$$

In this calculus, the $@\mu$ annotation corresponds to a delayed boxing of the term. In the var-rule, we compose the locks in the context with μ_2: as if μ_2 was itself a lock in the context. In return, the box rule does not have to put a lock into the context, and just adds ν to the $@\mu$ annotation of the term. The pair-rule pushes the delayed boxing into its components, encoding the implication $\Box_\mu A \times \Box_\mu B \to \Box_\mu (A \times B)$. In contrast, the lam-rule may not push the delayed boxing into the body of the lambda. Instead, it performs the boxing by adding a lock at μ to the context.

Adjoint Natural Deduction. Jang et al. [15] define a deduction system for tracking properties of values. They make the observation that their system models the usual IS4 modality as $\Box_{\mathrm{IS4}} A = \Box(\mathbb{1} \to A)$. This was also observed by Curien et al. [10], who similarly propose to split the exponential modality into an "proto-exponential whose promotion rule is restricted to values" and a suspension. These observations suggest that in order to track properties of computations, we can track properties of closure values instead.

Correspondence. In the appendix of the full version, 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 of the full version, we derive terms for $T : \Box A \to A$, $4 : \Box A \to \Box\,\Box A$, $X : \Box(A \to B) \times \Box(B \to C) \to \Box(A \to C)$, and $N : \Box\mathbb{1}$. Our calculus does not admit the rule of monotonicity ($\vdash A \to B$ implies $\vdash \Box A \to \Box B$), the K Axiom ($\Box(A \to B) \to (\Box A \to \Box B)$), 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 [4, 9, 24, 27].

Acknowledgments

We thank Ian Shillito and Nachi Valliappan for discussions on modal logic. Our intuition for the value trick was shaped by previous discussions with Stephen Dolan, Richard Eisenberg, and Leo White. 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] 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

[4] Francesco Berto and Daniel Nolan. 2021. Hyperintensionality. (2021).

[5] Chris Casinghino, Vilhelm Sjöberg, and Stephanie Weirich. 2014. Combining proofs and programs in a dependently typed language. *ACM SIGPLAN Notices* 49, 1 (2014), 33–45.

[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] Ranald 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] Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. 2016. A theory of effects and resources: adjunction models and polarised calculi. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg, FL, USA) *(POPL '16)*. Association for Computing Machinery, New York, NY, USA, 44–56. https://doi.org/10.1145/2837614.2837652

[11] 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

[12] Stephen Dolan and Leo White. 2022. Stack allocation for OCaml. Talk. In *OCaml workshop 2022*. ICFP.

[13] 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

[14] 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

[15] 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.

[16] Paulette Koronkevich and William J Bowman. 2025. Type Universes as Kripke Worlds. *Proceedings of the ACM on Programming Languages* 9, ICFP (2025), 784–813.

[17] Paul Blain Levy. 1999. Call-by-push-value: A subsuming paradigm. In *International Conference on Typed Lambda Calculi and Applications*. Springer, 228–243.

[18] 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

[19] 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.

[20] 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.

[21] 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

[22] 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.

[23] 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.

[24] Matteo Pascucci and Igor Sedlár. 2025. Hyperintensional models for non-congruential modal logics. *Logic Journal of the IGPL* 33, 5 (2025).

[25] 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

[26] Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Mathematical structures in computer science* 11, 4 (2001), 511–540.
[27] Igor Sedlár. 2021. Hyperintensional logics for everyone. *Synthese* 198, 2 (2021), 933–956.
[28] Wenhao Tang and Sam Lindley. 2026. Rows and Capabilities as Modal Effects. *Proceedings of the ACM on Programming Languages* 10, POPL (2026), 923–950.
[29] 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.
[30] 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.
[31] Philip Wadler. 1993. A taste of linear logic. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 185–210.