S4 modal sequent calculus as intermediate logic and intermediate language

JEAN CASPAR, École Normale Supérieure — PSL, INRIA, France
GUILLAUME MUNCH-MACCAGNONI, INRIA, LS2N CNRS, France

In this short paper, we advocate for the idea that continuation-based intermediate languages correspond to intermediate logics. The goal of intermediate languages is to serve as a basis for compiler intermediate representations, allowing to represent expressive program transformations for optimisation and compilation, while preserving the properties that make programs compilable efficiently in the first place, such as the "stackability" of continuations. Intermediate logics are logics between intuitionistic and classical logic in terms of provability.

Second-class continuations used in CPS-based intermediate languages correspond to a classical modal logic S4 with the added restriction that implications may only return modal types. This indeed corresponds to an intermediate logic, owing to the Gödel-McKinsey-Tarski theorem which states the intuitionistic nature of the modal fragment of S4.

We introduce $\mathbf{L}^{\Box}_{\mathrm{pol}}$, a three-kinded polarised sequent calculus for S4, together with an operational machine model that separates a heap from a stack. With this model we study a stackability property for the modal fragment of S4.

1 Introduction

Intermediate languages as intermediate logics. The proof-theoretical study of continuations through the question of constructivity in classical logic, via the formula-as-type correspondence between control operators and classical axioms, has been rich in lessons. It provided, for instance, a better understanding of control operators and better calculi to model computing with continuations, including the Curry-Howard correspondence between sequent calculus and abstract machines.

Historically, compilation has been one of the roots of the concept of continuation [Steele Jr. 1978]. Various works have applied the lessons of the proof-theoretical study to the construction of compilers [Downen et al. 2016; Schuster et al. 2025]. At the same time, there is a more general debate on the relevance of continuations for modern compilers [Maurer et al. 2017; Cong et al. 2019].

The notion of continuation used in the intermediate representations of compilers and the one studied by logicians turned out to be different. While the (polarised, *i.e.* evaluation-order aware, versions of) sequent calculi LJ (intuionistic: linear in the conclusions) and LK (classical: non-linear in the conclusions) correspond respectively to linearly-used and non-linear continuations, the restrictions on continuations that appear in intermediate representations seem to correspond—by Curry-Howard—to an intermediate logic between intuitionistic and classical sequent calculus. This point is made the most clearly by Downen, Maurer, Ariola, and Peyton Jones [2016]:

> [T]*he logic that comes out of Sequent Core lies in between* [LK and LJ]*: sometimes there can only be one conclusion, and sometimes there can be many. Furthermore* [...] *Sequent Core still captures a similar notion of purity from the λ-calculus. This demonstrates that there is room for more subtle variations on intuitionistic logic that lie between the freedom of LK and the purity of LJ.*

However, this intermediate character of continuations used for compilation remained at odds with the proof-theoretic treatment until Cong, Osvald, Essertel, and Rompf [2019], starting from an

January 13, 2026. Paper accepted for PEPM 2026 in the "short paper" category.
Authors' Contact Information: Jean Caspar, jean.caspar@ens.psl.eu, École Normale Supérieure — PSL, and INRIA, Paris, France; Guillaume Munch-Maccagnoni, guillaume.munch-maccagnoni@inria.fr, INRIA, LS2N CNRS, Nantes, France.

experiment to reproduce results from Maurer et al. [2017], proposed to capture the "second-class" character of continuations with a simple type system, based on ealier work on second-class values [Osvald, Essertel, Wu, González-Alayón, and Rompf 2016]. The language and type system from Cong et al. [2019] proved amenable indeed to be modelled via a sequent calculus for the modal logic S4 in the first author's MSc thesis [Caspar 2025], providing a logical characterisation of second-class values.

The calculus of Cong et al. [2019] distinguishes, thus, second-class (restricted) value from first-class (unrestricted) ones. The restrictions on second-class values ensure that they cannot escape from their defining scope: they cannot be returned from functions, and cannot be referred to from first-class values. In particular, a closure referring to a second-class value is itself second-class. This ensures [Osvald et al. 2016] that second-class values can be allocated on the stack, and that the stack can be freed when a function accepts a first-class argument. Continuations introduced by the compiler following Cong et al. [2019] are second class, ensuring they they can be allocated in a stack-like fashion. Nevertheless, from a Curry-Howard perspective, their CPS translation presents classical features: continuations can be duplicated or erased to a certain extent, relaxing the strict linear use of continuations in CPS models of intuitistic logic.

More recently, Schuster, Müller, Ostermann, and Brachthäuser [2025] challenged our preconceptions by using as a basis for efficient compilation a one-sided polarised classical sequent calculus—originally intended in Munch-Maccagnoni [2009] as a calculus for Girard's perfectly symmetric classical logic [Girard 1991; Danos, Joinet, and Schellinx 1997], and expressing unrestricted *call/cc* operators. Since it does not feature a priori restrictions on the use of continuations, the trick is to rely on a dynamic use of linearity in the back-end, using reference-counting with re-use of linearly-used memory cells. As they note, *"since continuation frames are typically used linearly, we hence get fast code for stack-like usage, even though we do not maintain a stack."*

We believe that there still are advantages in making the intermediate representations aware of the effective linearity of continuations, in order to better preserve (e.g. during code transformations for optimisation) and make use of (e.g. using hardware stack support) these properties which yield an efficient compilation strategy.

A classical S4 sequent calculus as an intermediate representation. Building upon the work by Cong, Osvald, Essertel, and Rompf [2019], this paper reports on some results from the first author's MSc thesis [Caspar 2025] which provided an investigation into second-class continuations with a polarised version of (classical) S4 modal sequent calculus, the calculus $\mathbf{L}^{\square}_{\text{pol}}$. One of its main results that motivates this short paper, though we do not detail it here, is its analysis of Cong et al. [2019] though the lens of modal logic:

> *The CPS translation in Cong et al. [2019] factors into an interpretation of their calculus into $\mathbf{L}^{\square}_{\text{pol}}$, followed by a CPS translation of $\mathbf{L}^{\square}_{\text{pol}}$.*

This extends a similar decomposition of CPS translations for the $\lambda\mu$-calculi via the one-sided classical sequent calculus [Danos et al. 1997; Ogata 2000; Laurent 2002; Munch-Maccagnoni 2013]. In this short paper, we omit the CPS translation for $\mathbf{L}^{\square}_{\text{pol}}$ and establish properties of its operational semantics directly.

The intuitionistic S4 modal logic has found a wide range of applications in programming languages, including staged compilation [Davies and Pfenning 2001], contextual type theory [Nanevski, Pfenning, and Pientka 2008], and information flow [Miyamoto and Igarashi 2004].

The *(classical)* S4 modal logic can be seen as mixing classical logic and intuitionistic logic, owing to the Gödel-McKinsey-Tarski theorem [Gödel 2001; McKinsey and Tarski 1948], according to which its modal fragment is complete for intuitionistic provability. Furthermore, if we consider a fragment of classical S4 modal logic obtained by restriction to functions whose return types are modal, we obtain an intermediate logic between classical and intuitionistic. We are indeed affecting the classical character of S4 with this restriction—the modal fragment is not altered—but limited forms of classical axioms are still provable. The intuition is that regular types are second class (cannot escape their defining scope, in programming terms), whereas modal types are first class.

A theory of second-class continuations via S4. In this paper, we present the polarised sequent calculus $\mathbf{L}^{\square}_{\mathrm{pol}}$ for the modal logic S4, with three polarities ($\square$, +, −). We also present a machine-like semantics with features similar to those of Cong et al. [2019]: we divide the memory into two parts, an ordinary heap for modal values, which can only refer to modal variables, and another part which is stack-like and gets freed every time we evaluate a modal covariable. The stack is not freed when returning a value with a non-modal type.

Then, if we restrict S4 to functions with a modal return type, thereby fully implementing the second-class restriction, we recover the stackability property of Cong et al. [2019]. The result suggests that metatheoretical results for second-class continuations follow in a straightforward manner from those for S4.

2 System $\mathbf{L}^{\square}_{\mathrm{pol}}$

In this section, we introduce a polarised sequent calculus for classical S4 logic with a term syntax called $\mathbf{L}^{\square}_{\mathrm{pol}}$, which extends the polarised classical L-calculus Munch-Maccagnoni [2009].

Brief recapitulation of the polarised L-calculus. The grammar of the calculus is formed by commands c of the form $\langle t \parallel e\rangle$, which are formed with an expression t in an environment e (a reification of an evaluation context). Environments can bind variables; for example, as usual, the program "let $x = t$ in c" can be represented as $\langle t \parallel \tilde{\mu}x.c\rangle$: it evaluates t and then executes c with the value of t being bound to the variable x.

Other constructs are available in $\mathbf{L}^{\square}_{\mathrm{pol}}$, for instance pairs (t, u) with pattern-matching on such pairs using the environment $\tilde{\mu}(x, y).c$. Thus, the program let $(x, y) = t$ in c is represented with $\langle t \parallel \tilde{\mu}(x, y).c\rangle$, which evaluates t to a value and then executes c with the value of t bound to (x, y), if this value indeed decomposes into a pair. We also have a unit value () on which we can match with $\tilde{\mu}().c$ and sum types with injections $\iota_i\, t$ and the match construct $\tilde{\mu}(\iota_1\, x.c \mid \iota_2\, y.c')$.

The type system corresponds to classical logic and follows the formulae-as-types interpretation for call/cc-style control operators [Griffin 1989]. The current continuation (or, here, the environment) is captured with the construction $\mu\alpha.c$: this expression captures the current environment, binds it to a "covariable" α, and continues with c.

Following the linear analysis of classical logic [Girard 1991; Danos et al. 1997], the apparent non-joinable critical pair, if α and x are fresh,

$$c \leftarrow \langle \mu\alpha.c \parallel \tilde{\mu}x.c'\rangle \rightarrow c'$$

is resolved by giving priority to the left-hand-side reduction for "positive" types (corresponding to CBV evaluation), and the right-hand-side reduction for "negative" types (corresponding to CBN evaluation). Binders and commands are annotated by a polarity ε.

We must also now distinguish which terms may be substituted or not. Thus, we have two other syntactic classes: values V and co-values S, which corresponds to "pure" expressions and environments that can be substituted. () is a value, and we restrict pairs and injections such that only (V, V') and $\iota_i\, V$ are allowed as values. $\mu\alpha^{\varepsilon}.c$ is a value when ε is negative, but not when ε is positive; $\tilde{\mu}x^{\varepsilon}.c$ is a co-value if ε is negative, but not when ε is negative. Except for those, every expression introduced before is a value, and every environment a co-value: variables are values, and covariables and $\tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c$ are co-values. We have the reduction rules:

$$\langle \mu\alpha^{\varepsilon}.c \parallel^{\varepsilon} S\rangle \to c[S/\alpha]$$
$$\langle V \parallel^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c\rangle \to c[V/x]$$

The reduction of $\langle \mu\alpha^{\varepsilon}.c \parallel^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c'\rangle$ is then determined uniquely by the polarity ε. Since the polarity is determined by the type, the polarities of both sides match if the command is well-typed.

This system supports both CBV and CBN, so in addition to strict pairs (V, W) we also have lazy records $\mu(\pi_1\ \alpha^{\varepsilon}.c \mid \pi_2\ \beta^{\varepsilon'}.c')$. This represents a record with two fields whose computation is not finished: to access a field, we can use the co-value $\pi_i\ S$, with S representing the environment in which we want to use the value computed from the field i. The reduction $\langle \mu(\pi_1\ \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\ \alpha_2^{\varepsilon_2}.c_2) \parallel^{-} \pi_i\ S\rangle \to c_i[S/\alpha_i]$ assigns this co-value to α_i and computes the field. We also have co-values representing pairs of co-values (S, S'), the corresponding values are $\mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c$, i.e. an expression which inspects the current co-value and retrieves S and S', and co-values consisting of a single value $[V]$, with opposing values $\mu[x^{\varepsilon}].c$, which implement negation.

Adding a polarity $\square$. The system presented so far corresponds to (a fragment of) classical polarised L-calculus. We do not merely add the $\square$ modality of S4: we consider a new polarity, $\square$, which is similar to + in the sense that it follows a CBV evaluation. Corresponding values are introduced by $\square V$, and co-values by $\tilde{\mu}\square x^{\varepsilon}.c$. Moreover, modal variables, annotated by $\square$, are special because a modal value can only refer to modal variables; this is enforced by typing.

What distinguishes a polarity from a modality is that a polarised type system reflects properties of stability under type constructions—typically, a positive pair of two modal values is itself modal. This is reflected in the type system with polarity tables which describe how the polarity of a type is deduced from its constituents.[1] In categorical terms, this amounts to shifting attention from the comonad $\square$ to its decomposition into an adjunction, typically with the category of $\square$-coalgebras, and interpeting a deductive system across several categories related by adjunctions. We will not get into categorical details here, but our types A with polarity $\square$ all enjoy $\square A \to A$.

In the end, we have three polarities: the usual positive and negative polarities $+, -$, and one additional modal positive polarity $\square$. Positive polarities $+, \square$ will be denoted $\boxplus$ and non-modal polarities $+, -$ will be denoted $\pm$. As we will see, the polarity $\square$ behaves almost like +, and shares a lot of constructs with it. However, their behaviour will differ in the second semantics we give in figs. 3 and 4.

ε	$::=$	$+ \mid - \mid \square$	*Polarities*
$\boxplus$	$::=$	$+ \mid \square$	*Positive polarities*
$\pm$	$::=$	$+ \mid -$	*Non-modal polarities*
V, W	$::=$	$x \mid (V, W) \mid \square V \mid () \mid \iota_1\ V \mid \iota_2\ V \mid \mu[x^{\varepsilon}].c$	*Values*
	$\mid$	$\mu(\pi_1\ \alpha^{\varepsilon_1}.c_1 \mid \pi_2\ \beta^{\varepsilon_2}.c_2) \mid \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \mid \mu\alpha^{-}.c$	
S, S'	$::=$	$\alpha \mid \pi_1\ S \mid \pi_2\ S \mid \tilde{\mu}\square x^{\varepsilon}.c \mid [V]$	*Co-values*
	$\mid$	$(S, S') \mid \tilde{\mu}().c \mid \tilde{\mu}x^{\boxplus}.c \mid \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c$	
	$\mid$	$\tilde{\mu}(\iota_1\ x^{\varepsilon_1}.c_1 \mid \iota_2\ y^{\varepsilon_2}.c_2)$	
t	$::=$	$V \mid \mu\alpha^{\boxplus}.c$	*Expressions*
e	$::=$	$S \mid \tilde{\mu}x^{-}.c$	*Environments*
c	$::=$	$\langle t \parallel^{\boxplus} S\rangle \mid \langle V \parallel^{-} e\rangle$	*Commands*

Grammar 1. Syntax of system $\mathbf{L}_{\mathrm{pol}}^{\square}$

[1]This idea goes back to Girard [1991, 1993], which indirectly influenced modal calculi via the Linear-Non-Linear term calculus and models [Benton 1994] (at least).

We define an operational semantics $\rhd$ for this system (fig. 1). It only reduces at the toplevel, and it is deterministic: if $c \rhd c_1$ and $c \rhd c_2$, then $c_1 = c_2$.

$$\begin{array}{ccc}
\langle \mu\alpha^{\varepsilon}.c \parallel^{\varepsilon} S \rangle & \rhd & c[S/\alpha] \\
\langle V \parallel^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c \rangle & \rhd & c[V/x] \\
\langle () \parallel^{\Box} \tilde{\mu}().c \rangle & \rhd & c \\
\langle (V, W) \parallel^{\varepsilon} \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c \rangle & \rhd & c[V/x, W/y] \\
\langle \iota_i\ V \parallel^{\varepsilon} \tilde{\mu}(\iota_1\ x_1^{\varepsilon_1}.c_1 \mid \iota_2\ x_2^{\varepsilon_2}.c_2) \rangle & \rhd & c_i[V/x_i] \\
\langle \Box V \parallel^{\Box} \tilde{\mu}\Box x^{\varepsilon}.c \rangle & \rhd & c[V/x] \\
\langle \mu[x^{\varepsilon}].c \parallel^{-} [V] \rangle & \rhd & c[V/x] \\
\langle \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \parallel^{-} (S, S') \rangle & \rhd & c[S/\alpha, S'/\beta] \\
\langle \mu(\pi_1\ \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\ \alpha_2^{\varepsilon_2}.c_2) \parallel^{-} \pi_i\ S \rangle & \rhd & c_i[S/\alpha_i]
\end{array}$$

Fig. 1. Operational semantics of system $\mathbf{L}^{\Box}_{\mathrm{pol}}$

3 Typing

Types and their associated polarities are defined as follows:

$$\underset{\varpi(A),}{A}\ ,\ \underset{\varpi(B)}{B}\ \underset{=}{::=}\ \underset{\Box}{\mathbb{1}} \mid \underset{x}{A \otimes B} \mid \underset{x}{A \oplus B} \mid \underset{\Box}{\Box A} \mid \underset{-}{\neg A} \mid \underset{-}{A \,\&\, B} \mid \underset{-}{A \parr B}$$

where $x = \varpi(A) \odot \varpi(B)$, defined further below. In words, to the connectives of classical linear logic without the exponentials, we add a modality $\Box$, representing a strong comonad. However, the system is not linear (hence the comonad is written "$\Box$" rather than "!"). Thus, even though $A \oplus B$ and $A \parr B$ are logically equivalent and represent a disjunction, they are not isomorphic and their reduction rules differ. Similarly, $\otimes$ and & both represent conjunction, and $\mathbb{1}$ is the unit of $\otimes$. Finally, $\neg$ is the (negative) negation.

Types have a polarity $\varpi(A) = \varepsilon$: we have $\varpi(A) = -$ for negative types, $\varpi(A) = +$ for positive types which are not modal, and $\varpi(A) = \Box$ for modal types; essentially coalgebras for $\Box$. Positive and modal types correspond to CBV types, and negative ones to CBN types. The operation $\odot$ on polarities is defined as follows:

$$\varepsilon \odot \varepsilon' = \begin{cases} \Box & \text{whenever } (\varepsilon, \varepsilon') = (\Box, \Box) \\ + & \text{otherwise.} \end{cases}$$

We note A_ε to assert that the type A is such that $\varpi(A) = \varepsilon$.

One can see that there is no function type in this system. The type $A \to B$, representing the arrow type of call-by-push-value, can be macro-defined as $\neg A \parr B$, with $\parr$ representing the negative disjunction; values of type $A \to B$ are introduced by $\mu(a \cdot \beta).c \stackrel{\text{def}}{=} \mu(\alpha, \beta).\langle \mu[a].c \parallel^{-} \alpha \rangle$ — in CBN, $\lambda x.t$ is defined as $\mu(x \cdot \beta).\langle t \parallel^{-} \beta \rangle$: the idea is that a value of type $A \to B$ captures a value of type A and a continuation of type B: the value the function returns must be sent to this continuation. Co-values of type $A \to B$ are introduced by $V \cdot S \stackrel{\text{def}}{=} ([V], S)$. The idea is that a co-value of type $A \to B$ consists of a value of type V on top of a co-value of type B, which represent the co-value on which the result of the function will run: for example, $t\ u$ is translated as $\mu\beta^{-}.\langle t \parallel^{-} u \cdot \beta \rangle$ (u is a value because in CBN, every expression is translated as a value of negative type).

Similarly, one can define polarity shifts, which cast the type so that it becomes positive/negative: the type $\Uparrow A \stackrel{\text{def}}{=} A \otimes \mathbb{1}$ is positive: its values are $\Uparrow V \stackrel{\text{def}}{=} (V, ())$ and its co-values are $\tilde{\mu}\Uparrow x^{\varepsilon}.c \stackrel{\text{def}}{=} \tilde{\mu}(x^{\varepsilon}, u^{\Box}).c$, and the type $\Downarrow A \stackrel{\text{def}}{=} \neg \mathbb{1} \parr A$ is negative, with values $\mu\Downarrow\alpha^{\varepsilon}.c \stackrel{\text{def}}{=} \mu(\beta^{-}, \alpha).c$ and co-values $\Downarrow S \stackrel{\text{def}}{=} ([()], S)$.

The modal logic S4 a logic which a modality $\square$ satisfying $\square(A \to B) \to \square A \to \square B$, $\square A \to A$ and $\square A \to \square\square A$. It also comes with the necessitation rule, saying that if A is provable without hypothesis then $\square A$ is provable without hypothesis. Here, $\square$ follows almost the same rules. However, mixing modalities and evaluation order (effects) reveals difficulties associated with a value restriction similarly to Curien, Fiore, and Munch-Maccagnoni [2016]: as they explain, having both a co-monad on a category of values, together with an effect adjunction, would require the rule $\vdash\square$ to be restricted to values. Thus, in $\mathbf{L}^{\square}_{\mathrm{pol}}$, from a proof of $A \to B$ we cannot in general deduce a proof of $\square A \to \square B$, but only $\square A \to \square {\Uparrow} B$, due to this value restriction. This peculiarity has been addressed differently in Curien et al. [2016] by taking the modality to be essentially $\square{\Uparrow}$. Our treatment of the modality differs from Curien et al. [2016] in two aspects: the construct $\square V$ does not introduce an implicit suspension, and modal contexts are handled as in Kavvos [2017]. Intuitively that $\square$ is product-preserving, as in the semantics of Kavvos [2017] and unlike the modality ! of Curien et al. [2016], for which our approach would not work.

Contexts $(\Gamma \mathrel{¦} \Theta \vdash \Delta)$ are made of three parts, represented by ordered lists of (co)variables: Γ, Θ and Δ. None of them are linear: they all supports permutation, contraction and weakening. Γ is the usual "intuitionistic" part of the context, and contains variables with their type. Θ is the modal part, also consisting of variables and their types: $x : A \in \Theta$ should be seen as $x : \square A \in \Gamma$, but it allows to extract the values behind the modality. Lastly, Δ is the classical part, consisting of co-variables $\alpha : A$, representing continuations. An empty list is represented as $\diamond$.

Indeed, following the Curry-Howard correspondence for classical logic, we know that formulas on the right-hand side of the $\vdash$ correspond to continuations variables. The syntax is essentially a fragment of the two-sided L-calculus Munch-Maccagnoni [2009, 2017], with a dual context on the left inspired by the syntax of Kavvos [2017] specific to product-preserving modalities. Lastly, sequents may also include a distinguished zone (except for commands judgments), which can be left or right, and contains at most one formula. Intuitively, the formula in this zone is the one we are working on.

We consider the typing rules given in fig. 2. There are five typing judgments:

- "$\Gamma \mathrel{¦} \Theta \vdash V : A; \Delta$" : V is a value of type A in the context $(\Gamma \mathrel{¦} \Theta \vdash \Delta)$;
- "$\Gamma \mathrel{¦} \Theta \vdash t : A \mid \Delta$" : t is an expression of type A in the context $(\Gamma \mathrel{¦} \Theta \vdash \Delta)$;
- "$\Gamma \mathrel{¦} \Theta; S : A \vdash \Delta$" : S is a co-value of type A in the context $(\Gamma \mathrel{¦} \Theta \vdash \Delta)$;
- "$\Gamma \mathrel{¦} \Theta \mid e : A \vdash \Delta$" : e is an environment of type A in the context $(\Gamma \mathrel{¦} \Theta \vdash \Delta)$;
- "$c : (\Gamma \mathrel{¦} \Theta \vdash \Delta)$" : c is a command in the context $(\Gamma \mathrel{¦} \Theta \vdash \Delta)$.

When writing $\Gamma, \Gamma', \Theta, \Theta'$ or Δ, Δ', we assume that Γ and Γ', Θ and Θ' and Δ and Δ' are disjoint. $\Gamma_{\square}$ denotes a context in which all types A are such that $\varpi(A) = \square$. We define renamings $\theta \in \mathfrak{R}(\Gamma \mathrel{¦} \Theta \vdash \Delta \Rightarrow \Gamma' \mathrel{¦} \Theta' \vdash \Delta')$: θ is a function between variables and covariables such that if $x : A \in \Gamma$ then $\theta(x) : A \in \Gamma'$, if $x : A \in \Theta$ then $\theta(x) : A \in \Theta'$ and if $\alpha : A \in \Delta$ then $\theta(\alpha) : A \in \Delta'$.

We note that some rule forces to separate the contexts, as the R$\otimes$ rule. However, this is not an issue, because we can use renamings to contract variables.

$$
\dfrac{\dfrac{}{x : A \mathrel{¦} \diamond \vdash x : A; \diamond}\,(\vdash\mathrm{Ax}) \qquad \dfrac{}{y : A \mathrel{¦} \diamond \vdash y : A; \diamond}\,(\vdash\mathrm{Ax})}{\dfrac{x : A, y : A \mathrel{¦} \diamond \vdash (x, y) : A \otimes A; \diamond}{z : A \mathrel{¦} \diamond \vdash (z, z) : A \otimes A; \diamond}\,(\vdash\mathfrak{R}(V))}\,(\vdash\otimes)
$$

We applied the renaming $\theta(x) = \theta(y) = z$. It preserves types : x and y are type A, and z too.

$$\frac{}{x : A \mathrel{¦} \diamond \vdash x : A; \diamond}\,(\vdash\text{Ax}) \qquad \frac{}{\diamond \mathrel{¦} x : A \vdash x : A; \diamond}\,(\vdash\Box\text{Ax})$$

$$\frac{}{\diamond \mathrel{¦} \diamond \mid \alpha : A \vdash \alpha : A}\,(\text{Ax}\vdash)$$

$$\frac{\Gamma \mathrel{¦} \Theta \vdash V : A_{\boxplus}; \Delta \qquad \Gamma' \mathrel{¦} \Theta' \mid e : A_{\boxplus} \vdash \Delta'}{\langle V \,\|^{\boxplus}\, e\rangle : (\Gamma, \Gamma' \mathrel{¦} \Theta, \Theta' \vdash \Delta, \Delta')}\,(\text{cut}\boxplus)$$

$$\frac{\Gamma \mathrel{¦} \Theta \vdash t : A_{-} \mid \Delta \qquad \Gamma' \mathrel{¦} \Theta'; S : A_{-} \vdash \Delta'}{\langle t \,\|^{-}\, S\rangle : (\Gamma, \Gamma' \mathrel{¦} \Theta, \Theta' \vdash \Delta, \Delta')}\,(\text{cut}-)$$

$$\frac{c : (\Gamma \mathrel{¦} \Theta \vdash \alpha : A_{\boxplus}, \Delta)}{\Gamma \mathrel{¦} \Theta \vdash \mu\alpha^{\boxplus}.c : A_{\boxplus} \mid \Delta}\,(\vdash\mu\boxplus) \qquad \frac{c : (\Gamma \mathrel{¦} \Theta \vdash \alpha : A_{-}, \Delta)}{\Gamma \mathrel{¦} \Theta \vdash \mu\alpha^{-}.c : A_{-}; \Delta}\,(\vdash\mu-)$$

$$\frac{c : (\Gamma, x : A_{\boxplus} \mathrel{¦} \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta; \tilde{\mu}x^{\boxplus}.c : A_{\boxplus} \vdash \Delta}\,(\tilde{\mu}\boxplus\vdash) \qquad \frac{c : (\Gamma, x : A_{-} \mathrel{¦} \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta; \tilde{\mu}x^{-}.c : A_{-} \vdash \Delta}\,(\tilde{\mu}-\vdash)$$

$$\frac{}{\diamond \mathrel{¦} \diamond \vdash () : \mathbb{1}; \diamond}\,(\vdash\mathbb{1}) \qquad \frac{c : (\Gamma \mathrel{¦} \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta; \tilde{\mu}().c : \mathbb{1} \vdash \Delta}\,(\mathbb{1}\vdash)$$

$$\frac{\Gamma \mathrel{¦} \Theta \vdash V : A; \Delta \qquad \Gamma' \mathrel{¦} \Theta' \vdash W : B; \Delta'}{\Gamma, \Gamma' \mathrel{¦} \Theta, \Theta' \vdash (V, W) : A \otimes B; \Delta, \Delta'}\,(\vdash\otimes)$$

$$\frac{c : (\Gamma, x : A_{\varepsilon_1}, y : B_{\varepsilon_2} \mathrel{¦} \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta; \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c : A_{\varepsilon_1} \otimes B_{\varepsilon_2} \vdash \Delta}\,(\otimes\vdash)$$

$$\frac{\Gamma \mathrel{¦} \Theta \vdash V : A_i; \Delta}{\Gamma \mathrel{¦} \Theta \vdash \iota_i\, V : A_1 \oplus A_2; \Delta}\,(\vdash\oplus)$$

$$\frac{c_1 : (\Gamma, x : A_{\varepsilon_1} \mathrel{¦} \Theta \vdash \Delta) \qquad c_2 : (\Gamma, y : B_{\varepsilon_2} \mathrel{¦} \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta; \tilde{\mu}(\iota_1\, x^{\varepsilon_1}.c_1 \mid \iota_2\, y^{\varepsilon_2}.c_2) : A_{\varepsilon_1} \oplus B_{\varepsilon_2} \vdash \Delta}\,(\oplus\vdash)$$

$$\frac{\Gamma \mathrel{¦} \Theta \vdash V : A_{\boxplus}; \Delta}{\Gamma \mathrel{¦} \Theta \vdash V : A_{\boxplus} \mid \Delta}\,(\vdash V) \qquad \frac{\Gamma \mathrel{¦} \Theta; S : A_{-} \vdash \Delta}{\Gamma \mathrel{¦} \Theta \mid S : A_{-} \vdash \Delta}\,(S\vdash)$$

$$\frac{c : (\Gamma \mathrel{¦} \Theta \vdash \alpha : A_{\varepsilon_1}, \beta : B_{\varepsilon_2}, \Delta)}{\Gamma \mathrel{¦} \Theta \vdash \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c : A_{\varepsilon_2} \parr B_{\varepsilon_1}; \Delta}\,(\vdash\parr)$$

$$\frac{\Gamma \mathrel{¦} \Theta \mid e : A \vdash \Delta \qquad \Gamma' \mathrel{¦} \Theta' \mid f : B \vdash \Delta'}{\Gamma, \Gamma' \mathrel{¦} \Theta, \Theta' \mid (e, f) : A \parr B \vdash \Delta, \Delta'}\,(\parr\vdash)$$

$$\frac{c : (\Gamma, x : A_{\varepsilon} \mathrel{¦} \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta \vdash \mu[x^{\varepsilon}].c : \neg A_{\varepsilon}; \Delta}\,(\vdash\neg) \qquad \frac{\Gamma \mathrel{¦} \Theta \vdash V : A; \Delta}{\Gamma \mathrel{¦} \Theta; [V] : \neg A \vdash \Delta}\,(\neg\vdash)$$

$$\frac{c_1 : (\Gamma, \mathrel{¦} \Theta \vdash \Delta, \alpha : A_{\varepsilon_1}) \qquad c_2 : (\Gamma, \mathrel{¦} \Theta \vdash \Delta, \beta : B_{\varepsilon_2})}{\Gamma \mathrel{¦} \Theta \vdash \mu(\pi_1\, \alpha^{\varepsilon_1}.c_1 \mid \pi_2\, \beta^{\varepsilon_2}.c_2) : A_{\varepsilon_1} \,\&\, B_{\varepsilon_2}; \Delta}\,(\vdash\&)$$

$$\frac{\Gamma \mathrel{¦} \Theta; S : A_i^{\varepsilon_i} \vdash \Delta}{\Gamma \mathrel{¦} \Theta; \pi_i\, S : A_1^{\varepsilon_1} \,\&\, A_2^{\varepsilon_2} \vdash \Delta}\,(\&\vdash)$$

$$\frac{\Gamma_{\Box} \mathrel{¦} \Theta \vdash V : A; \diamond}{\Gamma_{\Box} \mathrel{¦} \Theta \vdash \Box V : \Box A; \diamond}\,(\vdash\Box) \qquad \frac{c : (\Gamma \mathrel{¦} x : A_{\varepsilon}, \Theta \vdash \Delta)}{\Gamma \mathrel{¦} \Theta; \tilde{\mu}\Box x^{\varepsilon}.c : \Box A_{\varepsilon} \vdash \Delta}\,(\Box\vdash)$$

$$\frac{\Gamma \mathrel{¦} \Theta \vdash V : A; \Delta}{\Gamma' \mathrel{¦} \Theta' \vdash \theta(V) : A; \Delta'}\,(\vdash\Re(V)) \qquad \frac{\Gamma \mathrel{¦} \Theta \vdash t : A \mid \Delta}{\Gamma' \mathrel{¦} \Theta' \vdash \theta(t) : A; \Delta'}\,(\vdash\Re(t))$$

$$\frac{\Gamma \mathrel{¦} \Theta; S : A \vdash \Delta}{\Gamma' \mathrel{¦} \Theta'; \theta(S) : A \vdash \Delta'}\,(\Re(S)\vdash) \qquad \frac{\Gamma \mathrel{¦} \Theta \mid e : A \vdash \Delta}{\Gamma' \mathrel{¦} \Theta' \mid \theta(e) : A \vdash \Delta'}\,(\Re(e)\vdash)$$

$$\frac{c : (\Gamma \mathrel{¦} \Theta \vdash \Delta)}{\theta(c) : (\Gamma' \mathrel{¦} \Theta' \vdash \Delta')}\,(\Re(c))$$

Fig. 2. Typing rules of the system $\mathbf{L}^{\Box}_{\text{pol}}$

Similarly, axiom rules only allow one variable/covariable in the context. Again, this is not an issue:

$$\frac{\dfrac{}{\diamond \mathrel{¦} \diamond; \alpha : A \vdash \alpha : A}\,(\text{Ax}\vdash)}{\Gamma \mathrel{¦} \Theta; \alpha : A \vdash \alpha : A, \Delta}\,(\Re(S)\vdash)$$

Here we applied the renaming $\theta(\alpha) = \alpha$. Of course, it preserves the type of α. The other variables we added in the context do not appear in the expression, so nothing can go "wrong" with them.

Lastly, we can use renamings to exchange variables so that the context are separated at the right place, for example, here, with $\theta(x) = x$ and $\theta(y) = y$:

$$\frac{\dfrac{\dfrac{}{\diamond \mathrel{¦} y : B \vdash y : B; \diamond}\,(\vdash\Box\text{Ax}) \qquad \dfrac{}{\diamond \mathrel{¦} x : A \vdash x : A; \diamond}\,(\vdash\Box\text{Ax})}{\diamond \mathrel{¦} y : B, x : A \vdash (y, x) : B \otimes A; \diamond}\,(\vdash\otimes)}{\diamond \mathrel{¦} x : A; y : B \vdash (y, x) : B \otimes A; \diamond}\,(\vdash\Re(V))$$

This type system enjoys some soundness properties:

Lemma 3.1 (Modal restriction). *If V is a value and $\Gamma \mathrel{¦} \Theta \vdash V : A_{\Box}; \Delta$ then $\Gamma_{\Box} \mathrel{¦} \Theta \vdash V : A_{\Box}; \diamond$.*

This is the crucial property of this calculus. It ensures that modal values only refer to modal values in their closure; thus, if we allocate modal values on the heap and the rest on the stack, it ensures that values on the heap do not refer to the stack.

Lemma 3.2 (Typed substitution). *If c (or t, V, S or e) is a well-typed command, expression, etc. in the context $(\Gamma \mid \Theta \vdash \Delta)$, if $(\Gamma' \mid \Theta' \vdash \Delta')$ is another context and σ a function such that for all $x : A \in \Gamma$, $\sigma(x)$ is a value such that $\Gamma' \mid \Theta' \vdash \sigma(x) : A; \Delta'$, for each $x : A \in \Theta$, $\sigma(x)$ is a value such that $\Gamma'_{\square} \mid \Theta' \vdash \sigma(x) : A; \diamond$ and for each $\alpha : A \in \Delta$, $\sigma(\alpha)$ is a co-value such that $\Gamma' \mid \Theta'; \sigma(\alpha) : A \vdash \Delta'$, then $\sigma(c)$, respectively $\sigma(t)$, etc. is well-typed in the context $(\Gamma' \mid \Theta' \vdash \Delta')$. Moreover, if applicable, its type is the same.*

Lemma 3.3 (Subject reduction). *If $c \rhd c'$ and $c : (\Gamma \mid \Theta \vdash \Delta)$ then $c' : (\Gamma \mid \Theta \vdash \Delta)$.*

Lemma 3.4 (Normalization). *$\rhd$ is normalizing on well-typed terms.*

Proof. We have in fact the stronger result: the contextual closure of $\rhd$ is strongly normalizing. Indeed, $\mathbf{L}^{\square}_{\mathrm{pol}}$ can be translated into the LJ^{η}_{p} system from Curien et al. [2016] in a straightforward manner, by translating $\square A$ as $\Uparrow A$, $(\Gamma \mid \Theta \vdash \Delta)$ as $(\Gamma, \Uparrow\Theta \vdash \Delta)$, $\square V$ as $(V, ())$ and $\tilde{\mu}\square x.c$ as $\tilde{\mu}\Uparrow x.c$. Moreover, the polarity $\square$ is translated as $+$. This translation preserves typing and is a simulation, and their system is strongly normalizing (see e.g. [Munch-Maccagnoni 2017]); hence ours is also strongly normalizing. □

Together, these lemmas allow to prove that we can evaluate every "closed" command $c : (\diamond \mid \diamond \vdash tp : R_{\square})$ to a value, where R is a modal type and tp the toplevel continuation. Indeed, such a command reduces to a normal form c'. By case analysis, we can see that $c' = \langle V \parallel^{\square} tp \rangle$ for a certain V. This V has type R in the context $(\diamond \mid \diamond \vdash tp : R)$ by subject reduction, but thanks the modal restriction lemma, we can actually show that it contains no free variable. This V is unique because $\rhd$ is deterministic.

Theorem 3.5 (Evaluation). *If $c : (\diamond \mid \diamond \vdash tp : R_{\square})$ there exists a closed value V of type R such that $c \rhd^{*} \langle V \parallel^{\square} tp \rangle$. Moreover, this V is unique.*

This is interesting because in the end, we are interested in evaluating complete programs that return purely positive values like booleans or integers; which are indeed given by modal datatypes (e.g. $\mathbb{1} \oplus \mathbb{1}$ which is a modal type). Like the type system of $\lambda_{1/2}$ described in [Cong et al. 2019], in this system, we can wrap a type in $\square$ to force its values to be pure, ie., not to contain continuations, but we may locally use continuations in a function which returns a pure value, and the type system ensures that continuations do not escape.

4 Machine-like semantics

This section presents a machine-like semantics for the calculus which manages the memory explicitly. For their calculus, Cong et al. [2019] devised a machine with an interesting property, stackability: when calling a function that accepts a first-class argument, the stack gets freed and reset to how it was at the definition of the function. Here, we do not have the notion of first-class or second-class types directly, but a first-class type in their system corresponds to a modal type in ours, so a function accepting a first-class argument would have type $A_{\square} \to B_{\square} = \Uparrow(\neg A_{\square} \parr B_{\square})$ in ours. Note that B is always first-class in their setting, so modal in ours. The semantics we propose is based on generalizing the stackability property.

We first present the memory of our machine in gra. 2. It can store values and co-values. The typing rules will ensure that whenever a variable x is stored in $\mathcal{M}$, then it must be negative, ie. represent a thunk which has not yet been evaluated. More generally, if a positive or modal variable

x appears in a value V stored in the memory, then every occurence of x appears as a subterm of some command c, itself a subterm of V. Dually, negative covariables appearing in a co-value S stored inside $\mathcal{M}$ can only appear as subterms of some command which is a subterm of S. Thus, if x is positive, then the strict pair (x, x) can not be stored in $\mathcal{M}$, but $\mu(\pi_1\ \alpha^+.\langle x \parallel^+ \alpha\rangle \mid \pi_2\ \beta^+.\langle x \parallel^+ \beta\rangle)$, representing the lazy record both of whose fields have value x, can be stored in $\mathcal{M}$. Dually, stored covariables α must be positive or modal. To outline this distinction, we will write $\mathbb{V}$, $\mathbb{W}$ or $\mathbb{S}$ for values and co-values stored inside $\mathcal{M}$.

$$
\begin{array}{lcll}
\mathcal{H} & ::= & \diamond \quad | \quad \mathcal{H}, x^\varepsilon := \mathbb{V} & \textit{Heap} \\
\mathcal{E} & ::= & x^\pm := \mathbb{V} \quad | \quad \alpha^\varepsilon := \mathbb{S} & \textit{Equations} \\
\mathcal{F} & ::= & (\mathcal{E}) \quad | \quad (\mathcal{E}, \mathcal{E}) & \textit{Stack frame} \\
\mathcal{S} & ::= & \diamond \quad | \quad \mathcal{S}, \mathcal{F} & \textit{Stack} \\
\mathcal{M} & ::= & (\mathcal{H}; \mathcal{S}) & \textit{Memory}
\end{array}
$$

Grammar 2. Memory syntax

$\mathcal{M}$ is the memory; it is composed of a heap, $\mathcal{H}$, for values of modal types, and of a stack $\mathcal{S}$, for values of non-modal types and for co-values. We can also allocate values of non-modal type in $\mathcal{H}$, but only if they don't refer to variables outside of Θ except for variables of modal type. This ensures that values on the heap don't refer to the stack. The stack is divided into stack frames $\mathcal{F}$. We can assign new values with the operations $\mathcal{M}[x^\varepsilon := \mathbb{V}]$ and $\mathcal{M}[\alpha^\varepsilon := \mathbb{S}]$, which allocates the memory on the heap or the stack accordingly: each time new variables are allocated on the stack, a new stack frame is created. A stack frame has an address: we denote $@x$ or $@\alpha$ the adress of the stack frame in which x or α has been allocated. We can restrict the memory $\mathcal{M}|_{@\alpha}$, so that it drops the stack frame to which α belongs, and every stack frame allocated afterwards. We write $\mathcal{M}(x)$ and $\mathcal{M}(\alpha)$ for the value, respectively the stack, assigned to x and α inside $\mathcal{M}$, and $\varpi(x)$ or $\varpi(\alpha)$ for its polarity ε (values on the heap are all modal). The memory will be implicit in the notation.

More formally, for the allocation:

$$
\begin{array}{ccc}
(\mathcal{H};\mathcal{S})[x^\Box := \mathbb{V}] & \stackrel{\text{def}}{=} & (\mathcal{H}, (x^\Box := \mathbb{V}); \mathcal{S}) \\
(\mathcal{H};\mathcal{S})[x^\pm := \mathbb{V}] & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}, (x^\varepsilon := \mathbb{V})) \\
(\mathcal{H};\mathcal{S})[\alpha^\varepsilon := \mathbb{S}] & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}, (\alpha^\varepsilon := \mathbb{S})) \\
(\mathcal{H};\mathcal{S})[x^\Box := \mathbb{V}, y^\Box := \mathbb{V}'] & \stackrel{\text{def}}{=} & (\mathcal{H}, (x := \mathbb{V}), (y := \mathbb{V}'); \mathcal{S}) \\
(\mathcal{H};\mathcal{S})[x^\Box := \mathbb{V}, \alpha^\varepsilon := \mathbb{S}] & \stackrel{\text{def}}{=} & (\mathcal{H}, (x := \mathbb{V}); \mathcal{S}, (\alpha^\varepsilon := \mathbb{S})) \\
(\mathcal{H};\mathcal{S})[\alpha^\varepsilon := \mathbb{S}, \beta^{\varepsilon'} := \mathbb{S}'] & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}, (\alpha^\varepsilon := \mathbb{S}, \beta^{\varepsilon'} := \mathbb{S}')) \\
 & \text{etc.} &
\end{array}
$$

And for the restriction:

$$
\begin{array}{ccc}
(\mathcal{H};\diamond)|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\diamond) \\
(\mathcal{H};\mathcal{S}, (\alpha^\varepsilon := \mathbb{S}))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\mathcal{S}) \\
(\mathcal{H};\mathcal{S}, (\beta^\varepsilon := \mathbb{S}))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\mathcal{S})|_{@\alpha} \\
(\mathcal{H};\mathcal{S}, (x^\pm := \mathbb{V}))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\mathcal{S})|_{@\alpha} \\
(\mathcal{H};\mathcal{S}, (\beta^\varepsilon := \mathbb{S}, \gamma^{\varepsilon'} := \mathbb{S}'))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\mathcal{S})|_{@\alpha} \\
(\mathcal{H};\mathcal{S}, (\alpha^\varepsilon := \mathbb{S}, \beta^{\varepsilon'} := \mathbb{S}'))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\mathcal{S}) \\
(\mathcal{H};\mathcal{S}, (\beta^{\varepsilon'} := \mathbb{S}, \alpha^\varepsilon := \mathbb{S}'))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H};\mathcal{S}) \\
 & \text{etc.} &
\end{array}
$$

Now, we show how to evaluate values and co-values. We define the following relations (fig. 3):

$$\mathcal{M} \vDash V \Downarrow_v \mathbb{V} \qquad \mathcal{M} \vDash S \Downarrow_s \mathbb{S}$$

The first one means that V evaluates to the value $\mathbb{V}$ in the environment $\mathcal{M}$, and the second means that S evaluates to the co-value $\mathbb{S}$ in the environment $\mathcal{M}$. They do not force thunks; thus, negative variables are not evaluated, and similarly for positive or modal covariables.

$$\frac{\mathcal{M}(x) = \mathbb{V} \quad \varpi(x) \in \{+, \square\}}{\mathcal{M} \vDash x \Downarrow_v \mathbb{V}} \qquad \frac{\varpi(x) = -}{\mathcal{M} \vDash x \Downarrow_v x}$$

$$\frac{\varpi(\alpha) \in \{+, \square\}}{\mathcal{M} \vDash \alpha \Downarrow_s \alpha} \qquad \frac{\mathcal{M}(\alpha) = \mathbb{S} \quad \varpi(\alpha) = -}{\mathcal{M} \vDash \alpha \Downarrow_s \mathbb{S}}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V} \quad \mathcal{M} \vDash V' \Downarrow_v \mathbb{W}}{\mathcal{M} \vDash (V, V') \Downarrow_v (\mathbb{V}, \mathbb{W})} \qquad \frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\mathcal{M} \vDash \iota_i\, V \Downarrow_v \iota_i\, \mathbb{V}}$$

$$\frac{}{\mathcal{M} \vDash \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c \Downarrow_s \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c} \qquad \frac{\mathcal{M} \vDash S \Downarrow_s \mathbb{S}}{\mathcal{M} \vDash \pi_i\, S \Downarrow_s \pi_i\, \mathbb{S}}$$

$$\frac{}{\mathcal{M} \vDash \tilde{\mu}(\iota_1\, x_1^{\varepsilon_1}.c_1 \mid \iota_2\, x_2^{\varepsilon_2}.c_2) \Downarrow_s \tilde{\mu}(\iota_1\, x_1^{\varepsilon_1}.c_1 \mid \iota_2\, x_2^{\varepsilon_2}.c_2)}$$

$$\frac{}{\mathcal{M} \vDash \mu(\pi_1\, \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\, \alpha_2^{\varepsilon_2}.c_2) \Downarrow_v \mu(\pi_1\, \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\, \alpha_2^{\varepsilon_2}.c_2)}$$

$$\frac{}{\mathcal{M} \vDash () \Downarrow_v ()} \qquad \frac{}{\mathcal{M} \vDash \tilde{\mu}().c \Downarrow_s \tilde{\mu}().c}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\mathcal{M} \vDash [V] \Downarrow_s [\mathbb{V}]} \qquad \frac{}{\mathcal{M} \vDash \mu[x^{\varepsilon}].c \Downarrow_v \mu[x^{\varepsilon}].c}$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s \mathbb{S} \quad \mathcal{M} \vDash S' \Downarrow_s \mathbb{S}'}{\mathcal{M} \vDash (S, S') \Downarrow_s (\mathbb{S}, \mathbb{S}')} \qquad \frac{}{\mathcal{M} \vDash \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \Downarrow_v \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\mathcal{M} \vDash \square V \Downarrow_v \square \mathbb{V}} \qquad \frac{}{\mathcal{M} \vDash \tilde{\mu}\square x^{\varepsilon}.c \Downarrow_s \tilde{\mu}\square x^{\varepsilon}.c}$$

Fig. 3. Evaluation of values and stacks

For example, we have:

$$(x \coloneqq (); (y^- \coloneqq \mathbb{V})) \vDash (x, y) \Downarrow_v ((), y)$$

Now, we define in fig. 4 the relation $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$, which means that with the memory $\mathcal{M}$, the command c reduces to the command c' and modifies the memory to be $\mathcal{M}'$. The key point of this semantics are the rule Eval$\tilde{\mu}\square$, which allocates modal variables (belonging to Θ) on the heap:

$$\frac{\mathcal{M} \vDash V \Downarrow_v \square \mathbb{V}}{((\mathcal{H}; \mathcal{S}) \vDash \langle V \parallel^{\square} \tilde{\mu}\square x^{\varepsilon}.c\rangle) \rightsquigarrow ((\mathcal{H}, (x^{\varepsilon} \coloneqq \mathbb{V}); \mathcal{S}) \vDash c)}\ \text{Eval}\tilde{\mu}\square$$

and the rule Eval$\square$:

$$\frac{\mathcal{M}(\alpha) =^{\square} \mathbb{S}}{(\mathcal{M} \vDash \langle V \parallel^{\square} \alpha\rangle) \rightsquigarrow (\mathcal{M}|_{@\alpha} \vDash \langle V \parallel^{\square} \mathbb{S}\rangle)}\ \text{Eval}\square$$

$$\frac{\mathcal{M}(\alpha) =^{+} \mathbb{S}}{(\mathcal{M} \vDash \langle V \|^{+} \alpha\rangle) \rightsquigarrow (\mathcal{M} \vDash \langle V \|^{+} \mathbb{S}\rangle)}\ \text{Eval}+ \qquad \frac{\mathcal{M}(x) =^{-} \mathbb{V}}{(\mathcal{M} \vDash \langle x \|^{-} S\rangle) \rightsquigarrow (\mathcal{M} \vDash \langle \mathbb{V} \|^{-} S\rangle)}\ \text{Eval}-$$

$$\frac{\mathcal{M}(\alpha) =^{\square} \mathbb{S}}{(\mathcal{M} \vDash \langle V \|^{\square} \alpha\rangle) \rightsquigarrow (\mathcal{M}|_{@\alpha} \vDash \langle V \|^{\square} \mathbb{S}\rangle)}\ \text{Eval}\square \qquad \frac{\mathcal{M} \vDash S \Downarrow_s [\mathbb{V}]}{(\mathcal{M} \vDash \langle \mu[x^{\varepsilon}].c \|^{-} S\rangle) \rightsquigarrow (\mathcal{M}[x^{\varepsilon} := \mathbb{V}] \vDash c)}\ \text{Eval}\mu\neg$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s \mathbb{S}}{(\mathcal{M} \vDash \langle \mu\alpha^{\varepsilon}.c \|^{\varepsilon} S\rangle) \rightsquigarrow (\mathcal{M}[\alpha^{\varepsilon} := \mathbb{S}] \vDash c)}\ \text{Eval}\mu\alpha^{\varepsilon} \qquad \frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{(\mathcal{M} \vDash \langle V \|^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c\rangle) \rightsquigarrow (\mathcal{M}[x^{\varepsilon} := \mathbb{V}] \vDash c)}\ \text{Eval}\tilde{\mu}x^{\varepsilon}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v (\mathbb{V}, \mathbb{V}')}{(\mathcal{M} \vDash \langle V \|^{\varepsilon} \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c\rangle) \rightsquigarrow (\mathcal{M}[x^{\varepsilon_1} := \mathbb{V}, y^{\varepsilon_2} := \mathbb{V}'] \vDash c)}\ \text{Eval}\tilde{\mu}\otimes$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s (\mathbb{S}, \mathbb{S}')}{(\mathcal{M} \vDash \langle \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \|^{-} S\rangle) \rightsquigarrow (\mathcal{M}[\alpha^{\varepsilon_1} := \mathbb{S}, \beta^{\varepsilon_2} := \mathbb{S}'] \vDash c)}\ \text{Eval}\mu⅋$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \iota_i\, \mathbb{V}}{\left(\mathcal{M} \vDash \langle V \|^{\varepsilon} \tilde{\mu}(\iota_1\, x_1^{\varepsilon_1}.c_1 \mid \iota_2\, x_2^{\varepsilon_2}.c_2)\rangle\right) \rightsquigarrow \left(\mathcal{M}[x_i^{\varepsilon_i} := \mathbb{V}] \vDash c_i\right)}\ \text{Eval}\tilde{\mu}\oplus$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s \pi_i\, \mathbb{S}}{\left(\mathcal{M} \vDash \langle \mu(\pi_1\, \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\, \alpha_2^{\varepsilon_2}.c_2) \|^{-} S\rangle\right) \rightsquigarrow \left(\mathcal{M}[\alpha_i^{\varepsilon_i} := \mathbb{S}] \vDash c_i\right)}\ \text{Eval}\mu\&$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v ()}{(\mathcal{M} \vDash \langle V \|^{\square} \tilde{\mu}().c\rangle) \rightsquigarrow (\mathcal{M} \vDash c)}\ \text{Eval}\tilde{\mu}\mathbb{1} \qquad \frac{\mathcal{M} \vDash V \Downarrow_v \square\mathbb{V}}{((\mathcal{H};\mathcal{S}) \vDash \langle V \|^{\square} \tilde{\mu}\square x^{\varepsilon}.c\rangle) \rightsquigarrow ((\mathcal{H}, x^{\varepsilon} := \mathbb{V};\mathcal{S}) \vDash c)}\ \text{Eval}\tilde{\mu}\square$$

Fig. 4. Evaluation of commands

After evaluating a covariable α of modal type, we can make the stack shrink. Indeed, if we restrict ourselves to typed values, V is of modal type, so it cannot contains references to $\mathcal{S}$; and the value $\mathbb{S}$ of α must be typed in the environment we had before α was allocated.

Let us consider what happens when evaluating, for example, a (CBV) function t on some expression u. We suppose that t has for type $\Uparrow(A_{\square} \to B_{\square})$, that is, where A and B are modal. Note that t is not of modal type. This is the situation considered in Cong et al. [2019]. Now consider the evaluation of the following expression:

$$\mu\gamma^{\square}.\left\langle t \,\middle\|^{+}\, \tilde{\mu}f^{+}.\left\langle \mu\alpha^{\square}.\langle u \|^{\square} \alpha\rangle \,\middle\|^{\square}\, \tilde{\mu}x^{\square}.\langle f^{+} \|^{+} \tilde{\mu}{\Uparrow}g^{-}.\langle g \|^{-} x \cdot \gamma\rangle\rangle\right\rangle\right\rangle$$

First, we must evaluate t. Suppose it evaluates to f. Then we evaluate u (note the η-expansion). Suppose it returns a value V: this value will be evaluated against α, appearing in $\langle V \|^{\square} \alpha\rangle$. Thus, at this point, because α is modal, the stack will be reset a first time to the point where it was before we started evaluating u. This is valid because V contains no reference to the stack, as it is modal, and α points to $\tilde{\mu}x^{\square}.\langle f^{+} \|^{+} \tilde{\mu}{\Uparrow}g^{-}.\langle g \|^{-} x \cdot \gamma\rangle\rangle$, whose free variables have been allocated before the beginning of the evaluation of u. Then, we store $x := V$ on the heap, we extract the real function g from its wrapper f and evaluate it on x. This returns some value V', which is evaluated against γ. At this point, because γ is also modal, we can restore the stack to how it was before the call of $t\ u$: thus, the stack is restored in its initial state, and no memory has been leaked. Even if A is non-modal, we can still ensure that the stack is freed when the function returns, though we cannot free it after evaluating u. This is exactly the stack discipline of the CPS translation presented in Cong et al. [2019].

More generally, in this semantics, co-values S and non-modal values V follow a stack discipline:

Theorem 4.1 (Stackability). *The semantics that allocates co-values S and non-modal values V on the stack $\mathcal{S}$, shrinking every time the evaluation of a modal expression returns, bringing the stack back in its state before the evaluation of the modal expression started, is sound.*

In particular, this applies to functions whose return type is modal, and to their arguments if they are modal. That means that the stack in which we allocate correspond to the call stack if we restrict the calculus to functions whose return type is modal.

However, the behaviour of functions returning non-modal types is a bit weird: for example, $(\lambda x^+.\,\text{let } y = (x,x) \text{ in } y)\ V$ is translated as $t \stackrel{\text{def}}{=} \mu\beta^+.\big\langle \mu(x\cdot\alpha).\langle (x,x) \parallel^+ \tilde{\mu}y^+.\langle y \parallel^+ \alpha\rangle\rangle \big\Vert^- V\cdot\beta\big\rangle$, for which we have:

$$((\mathcal{H};\mathcal{S}) \vDash \langle t \parallel^+ S\rangle) \quad \rightsquigarrow^* \quad ((\mathcal{H};\mathcal{S},(\beta^+ \coloneqq \mathbb{S}),(x^+ \coloneqq \mathbb{V},\alpha^+ \coloneqq \beta),(y^+ \coloneqq (x,x))) \vDash \langle y \parallel^+ \alpha\rangle)$$

where S is an arbitrary co-value of positive type, and where we have $(\mathcal{H};\mathcal{S}) \vDash S \Downarrow_s \mathbb{S}$ and $(\mathcal{H};\mathcal{S},(\beta^+ \coloneqq \mathbb{S})) \vDash V \Downarrow_v \mathbb{V}$. That is, the returned value, y depends on the last stack frame, especially on the binding $y^+ \coloneqq (x,x)$, which was not allocated at the time the evaluation started. Thus the stack cannot shrink and return to $\mathcal{S}$ at this point of the execution.

In particular, this means that without the full second-class restriction, when functions can return non-modal values, the stack in which we allocate does not correspond to a call stack. On the one hand, whether and how the stackability property allows us to improve over indiscriminate heap-allocation remains to be investigated. On the other hand, this is not necessary to our point that the metatheory of second-class continuations follows from that of S4 sequent calculus—and the latter might admit more relaxed characterisations of the notion of non-escaping values (in the same way as second-class values are a simplistic form of borrowing).

5 Correctness

We now prove the correctness of this machine-like semantics with respect to the operational semantics $\triangleright$.

We first define a typing relation for environments in fig. 5. We first fix a modal type R, which will not change along this work, and will serve as a return type, and the type of tp. Type environments are defined almost like environments, except that we replace $x^\varepsilon \coloneqq \mathbb{V}$ by $x : A_\varepsilon$, etc. We note such a type environment Σ or $(\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}})$. We can associate a typing context to a type environment: variables in $\mathcal{S}$ go into Γ, covariables go into Δ and variables in $\mathcal{H}$ go into Θ. We also always add $tp : R$ to Δ. For example,

$$[\![((x:A),(y:B);(\alpha:C,\beta:D),(z:E))]\!] = z:E \mid x:A, y:B \vdash tp:R, \alpha:C, \beta:B$$

Note that our rules ensure that a value or a stack allocated on the stack only contains references to the heap and to variables allocated before. Theorem 3.1 ensures that values on the heap only refer to the heap.

We enumerate some useful properties of this type system:

Lemma 5.1 (Progress). *If $\mathcal{M} : \Sigma$ and c is well-typed in the environment $[\![\Sigma]\!]$ then $c = \langle \mathbb{V} \parallel^+ tp\rangle$ or there exists c' and $\mathcal{M}'$ such that $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$.*

Lemma 5.2 (Subject reduction). *If $\mathcal{M} : \Sigma$ and c is well-typed in the environment $[\![\Sigma]\!]$ and $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$, then $\mathcal{M}'$ is typable, either in an extension or in a restriction Σ' of Σ, and c' is well-typed in the environment Σ'.*

Proof sketch. By case analysis on the derivation $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$. For the case of the rule *Eval*$\Box$, we use the theorem 3.1 to prove that V contains no reference to $\mathcal{S}$; and then, if $\mathcal{M}(\alpha) =^\Box \mathbb{S}$ and $\mathcal{M}$ is well-typed, then $\mathbb{S}$ must be typed in $[\![\mathcal{M}|_{@\alpha}]\!]$. Thus $\langle V \parallel^\Box \mathbb{S}\rangle$ is well-typed in $[\![\mathcal{M}|_{@\alpha}]\!]$. □

$$\frac{}{(\diamond;\diamond) : (\diamond;\diamond)}$$

$$\frac{(\mathcal{H};\mathcal{S}) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}) \qquad \Gamma_{\square} \mathrel{¦} \Theta \vdash \mathbb{V} : A_{\varepsilon};\diamond \qquad [\![(\mathcal{H};\mathcal{S})]\!] = (\Gamma \mathrel{¦} \Theta \vdash \Delta)}{(\mathcal{H}, (x^{\varepsilon} \coloneqq \mathbb{V});\mathcal{S}) : (\Sigma_{\mathcal{H}}, (x : A_{\varepsilon});\Sigma_{\mathcal{S}})}$$

$$\frac{(\mathcal{H};\mathcal{S}) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}) \qquad \Gamma \mathrel{¦} \Theta \vdash \mathbb{V} : A_{\pm};\Delta \qquad [\![(\mathcal{H};\mathcal{S})]\!] = (\Gamma \mathrel{¦} \Theta \vdash \Delta)}{(\mathcal{H};\mathcal{S}, (x^{\pm} \coloneqq \mathbb{V})) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}, (x : A_{\pm}))}$$

$$\frac{(\mathcal{H};\mathcal{S}) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}) \qquad \begin{array}{c}\Gamma \mathrel{¦} \Theta;\mathbb{S} : A_{\varepsilon} \vdash \Delta \\ \Gamma \mathrel{¦} \Theta;\mathbb{S}' : B_{\varepsilon'} \vdash \Delta\end{array} \qquad [\![(\mathcal{H};\mathcal{S})]\!] = (\Gamma \mathrel{¦} \Theta \vdash \Delta)}{(\mathcal{H};\mathcal{S}, (\alpha^{\varepsilon} \coloneqq \mathbb{S}, \beta^{\varepsilon'} \coloneqq \mathbb{S}')) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}, (\alpha : A_{\varepsilon}, \beta : B_{\varepsilon'}))}$$

etc.

Fig. 5. Typing of environments

Lemma 5.3. *If $\mathcal{M} : \Sigma$ then $\mathcal{M}$ defines a typed substitution from $[\![\Sigma]\!]$ to $[\![(\diamond;\diamond)]\!]$, which will be noted* $[\mathcal{M}]$.

Lemma 5.4. *If $\mathcal{M} : \Sigma$ and V has type A in the environment $[\![\Sigma]\!]$ and $\mathcal{M} \vDash V \Downarrow_v \mathbb{V}$ then $V[\mathcal{M}] = \mathbb{V}[\mathcal{M}]$; the same goes for $\mathbb{S}$.*

Lemma 5.5. *If $\mathcal{M} : \Sigma$ and c is well-typed in the environment $[\![\Sigma]\!]$ and $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$ then $c[\mathcal{M}] \rhd^{\leq 1} c'[\mathcal{M}']$. Moreover, there cannot be an infinite sequence of $\rightsquigarrow$ that yields an equality.*

Now, if c is closed, ie. defined in the environment $[\![(\diamond;\diamond)]\!] = (\diamond \mathrel{¦} \diamond \vdash tp : R)$, by theorem 3.5 there exists a unique V of type R such that $c \rhd^* \langle V \Vert^{+} tp\rangle$. We will show the following:

Theorem 5.6 (Evaluation). *There exists V' and $\mathcal{H}$ such that $((\diamond;\diamond) \vDash c) \rightsquigarrow^* ((\mathcal{H};\mathcal{S}) \vDash \langle V' \Vert^{\square} tp\rangle)$, V' contain no reference to $\mathcal{S}$ and $V'[\mathcal{H}] = V$.*

This proves the soundness of our semantics.

Proof sketch. With the lemmas proved before, we can show that $\rightsquigarrow$ is normalizing. Moreover, at the end, the resulting command must be of the form $\langle V' \Vert^{\square} tp\rangle$, in some environment $\mathcal{M}'$. But V' is a value of modal type, so it cannot contain references to any variables or covariables in $\mathcal{S}'$ or to tp, by theorem 3.1. ◻

References

Nick Benton. 1994. "A mixed linear and non-linear logic: proofs, terms and models." In: *Proc. CSL* (Lecture Notes in Computer Science). Vol. 933. Springer-Verlag.

Jean Caspar. Aug. 20, 2025. "Mixing classical and intuitionistic constructiveness: a formulae-as-types interpretation for classical S4 modal logic and second-class continuations." Master's thesis. INRIA, (Aug. 20, 2025).

Youyou Cong, Leo Osvald, Grégory M. Essertel, and Tiark Rompf. July 26, 2019. "Compiling with continuations, or without? whatever." *Proceedings of the ACM on Programming Languages*, 3, (July 26, 2019), 1–28, ICFP, (July 26, 2019). doi:10.1145/3341643.

Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. Jan. 11, 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.* POPL '16: The 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. ACM, St. Petersburg FL USA, (Jan. 11, 2016), 44–56. isbn: 978-1-4503-3549-2. doi:10.1145/2837614.2837652.

Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. 1997. "A New Deconstructive Logic: Linear Logic." *The Journal of Symbolic Logic*, 62, 3, 755–807. doi:10.2307/2275572.

Rowan Davies and Frank Pfenning. May 2001. "A modal analysis of staged computation." *J. ACM*, 48, 3, (May 2001), 555–604. doi:10.1145/382780.382785.

Paul Downen, Luke Maurer, Zena M. Ariola, and Simon Peyton Jones. Sept. 4, 2016. "Sequent calculus as a compiler intermediate language." *SIGPLAN Not.*, 51, 9, (Sept. 4, 2016), 74–88. doi:10.1145/3022670.2951931.

Jean-Yves Girard. Nov. 1991. "A new constructive logic: classic logic." *Mathematical Structures in Computer Science*, 1, 3, (Nov. 1991), 255–296. doi:10.1017/s0960129500001328.

Jean-Yves Girard. Feb. 16, 1993. "On the unity of logic." *Annals of Pure and Applied Logic*, 59, 3, (Feb. 16, 1993), 201–217. doi:10.1016/0168-0072(93)90093-S.

Kurt Gödel. July 5, 2001. "An interpretation of the intuitionistic prepositional calculus (1933f)." In: *Kurt Gödel: Collected Works: Volume I: Publications 1929-1936*. Ed. by Kurt Gödel et al. Oxford University Press, Oxford, New York, (July 5, 2001). ISBN: 978-0-19-514720-9.

Timothy G. Griffin. Dec. 1, 1989. "A formulae-as-type notion of control." In: *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (POPL '90). Association for Computing Machinery, New York, NY, USA, (Dec. 1, 1989), 47–58. ISBN: 978-0-89791-343-0. doi:10.1145/96709.96714.

G. A. Kavvos. June 2017. "Dual-context calculi for modal logic." In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). IEEE, Reykjavik, Iceland, (June 2017), 1–12. ISBN: 978-1-5090-3018-7. doi:10.1109/LICS.2017.8005089.

Olivier Laurent. Mar. 2002. "Etude de la polarisation en logique." Thèse de Doctorat. Université Aix-Marseille II, (Mar. 2002).

Luke Maurer, Paul Downen, Zena M. Ariola, and Simon Peyton Jones. June 2017. "Compiling without continuations." *SIGPLAN Not.*, 52, 6, (June 2017), 482–494. doi:10.1145/3140587.3062380.

J. C. C. McKinsey and Alfred Tarski. 1948. "Some Theorems About the Sentential Calculi of Lewis and Heyting." *The Journal of Symbolic Logic*, 13, 1, 1–15. doi:10.2307/2268135.

Kenji Miyamoto and Atsushi Igarashi. 2004. "A modal foundation for secure information flow." In: *Workshop on Foundations of Computer Security*, 187–203.

Guillaume Munch-Maccagnoni. 2009. "Focalisation and Classical Realisability (version with appendices)." In: *Proceedings of the 18th EACSL Annual Conference on Computer Science Logic - CSL 09*. Vol. 5771. Springer-Verlag, 409. doi:10.1007/978-3-642-04027-6_30.

Guillaume Munch-Maccagnoni. 2017. *Note on Curry's style for Linear Call-by-Push-Value*. (2017). https://inria.hal.science/hal-01528857v3.

Guillaume Munch-Maccagnoni. Dec. 10, 2013. "Syntax and Models of a non-Associative Composition of Programs and Proofs." Ph.D. Dissertation. Université Paris-Diderot - Paris VII, (Dec. 10, 2013). https://theses.hal.science/tel-00918642.

Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. June 2008. "Contextual modal type theory." *ACM Trans. Comput. Logic*, 9, 3, Article 23, (June 2008), 49 pages. doi:10.1145/1352582.1352591.

Ichiro Ogata. 2000. "Constructive classical logic as CPS-calculus." *International Journal of Foundations of Computer Science*, 11, 01, 89–112. eprint: https://doi.org/10.1142/S0129054100000065. doi:10.1142/S0129054100000065.

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." In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. OOPSLA 2016. Association for Computing Machinery, New York, NY, USA, 234–251. ISBN: 978-1-4503-4444-9. doi:10.1145/2983990.2984009.

Philipp Schuster, Marius Müller, Klaus Ostermann, and Jonathan Immanuel Brachthäuser. Apr. 9, 2025. "Compiling Classical Sequent Calculus to Stock Hardware: The Duality of Compilation." *Proceedings of the ACM on Programming Languages*, 9, (Apr. 9, 2025), 1746–1773, OOPSLA1, (Apr. 9, 2025). doi:10.1145/3720507.

Guy L. Steele Jr.. May 1978. "RABBIT: A Compiler for SCHEME." Master's thesis. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, (May 1978). Technical report AI-TR-474.