A Stateful Time-Aware Operational Semantics for Temporal Resources

Danel Ahman
Institute of Computer Science, University of Tartu, Estonia
danel.ahman@ut.ee

Gašper Žajdela
Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
gasper.zajdela@fmf.uni-lj.si

By temporal resources we mean resources in programming whose usage is time-sensitive or -critical, and which, while perhaps already physically available to a program, can be used or acted upon only after a certain amount of time passes or some prescribed events take place. It has been shown that such temporal aspects of resources can be modularly captured with a combination of graded modal types and a graded-monadic effect system. In this paper, we develop this line of work further, by accompanying this modal approach with a stateful time-aware operational semantics, which in addition to explaining how programs behave computationally, can be used to analyse their run times and also which temporal resources they operate with. In addition to basic type-safety, we prove our semantics sound with respect to a temporal variant of the equational presentation of local state.

1 Introduction

The proper behaviour of programs usually depends on programmers correctly accessing and using resources in the programs that they write. The allowed or required usage-patterns can often be specified and checked using type systems and appropriately designed programming abstractions. For instance, this might mean that the correct operation of a program requires one to use some resources (e.g., a file pointer) linearly or uniquely [4, 6, 12, 24], some resources (e.g., a network channel) according to some prescribed protocol specification [8, 28], and some resources in such a way that they always get finalised [2, 14].

Recently, the first author showed that in addition to controlling *how* programs use their resoruces, type systems and carefully designed programming abstractions can be used to specify and check *when* programs are allowed to use their resources in a mathematically structured and compositional way [1]. This approach employs time-graded modal types to specify when a resource can be accessed or used by the program, importantly, also accounting for situations where the program might be physically in possession of the resource long before it is allowed to safely use it. Such modal type-based specifications are then combined with a graded-monadic effect system that is used to model time-passage at the type system level, and to make it possible to access further such *temporal resources* as more time is spent in a given program, either by explicitly blocking execution or by instead performing some useful work.

For example, by giving the operations disassemble, clean, primer, paint, and assemble output types specifying how much time has to pass before another operation can use the resources returned by these operations, the following code snippet typechecks in this modal approach only *when enough time is spent*

Submitted to:
MSFP 2024

in place of the comments, e.g., by working on assembling products on an adjacent production line:

```
let (partA, partB, partC) = disassemble product in
(* no time has to pass before using partA, partB, partC below *)
let (partB', partC') = clean (partB, partC) in
(* cleaning liquid has to vaporise before using partB', partC' below *)
let (partB'', partC'') = primer (partB', partC') in
(* primer has to dry before using partB'', partC'' below *)
let (partB''', partC''') = paint (partB'', partC'') in
(* paint has to dry before using partB''', partC''' below *)
let product' = assemble (partA, partB''', partC''') in
(* no time has to pass before using the assembled product' in the rest of the code *)
...
```

In the above-mentioned work, these ideas are formalised in a modally typed, effectful core λ-calculus, called $\lambda_{[\tau]}$, which is equipped with a simple equational theory, and a presheaves and graded monads based denotational semantics. In this paper, we develop this line of work further, by accompanying this modal approach with an operational semantics, which in addition to explaining how programs behave computationally, can be used to analyse their run times and which resources they operate with.

The paper is structured as follows: In Sect. 2, we review the $\lambda_{[\tau]}$-calculus introduced by the first author for working with temporal resources and propose minor modifications to make it amenable to an operational treatment. We also improve the treatment of variable renamings in $\lambda_{[\tau]}$ compared to the original paper. In Sect. 3, we define a stateful time-aware operational semantics for $\lambda_{[\tau]}$ and prove it type-safe. In Sect. 4, we further prove this semantics sound with respect to a temporal variant of the equational theory of local state. We then review related work in Sect. 5 and conclude in Sect. 6.

We have formalised the variant of $\lambda_{[\tau]}$ considered in this paper also in Agda [27], available at https://github.com/rzgdmqt/temporal-resources/releases/tag/msfp-2024-submission. The formalisation includes the definition of $\lambda_{[\tau]}$, renamings and substitutions, progress and preservation theorems, including necessary auxiliary results, and the equational theory defined in Sect. 4.2. It however lacks the proof of equational soundness (Thm. 4.9), whose formalisation remains work in progress, specifically due to the high number of renamings and substitutions applied to different variants of terms and term contexts.

2 Core Calculus for Temporal Resources

We begin by reviewing the $\lambda_{[\tau]}$-calculus introduced by the first author [1]. It is designed to capture the essence of type-based specification of and programming with temporal resources. As noted above, we also introduce minor modifications to the calculus to make it amenable to an operational treatment.

2.1 Terms

The $\lambda_{[\tau]}$-calculus is based on Levy et al.'s [16, 17] fine-grain call-by-value (FGCBV) calculus, extending it with time-graded modal types, temporally aware algebraic effects and handlers, and an effect system.

Being based on FGCBV, $\lambda_{[\tau]}$ distinguishes between *values* and *computations*, defined in Fig. 1. Values are standard from FGCBV: variables, constants, unit, pairing, and functions. Computations are also largely standard from FGCBV: they include returning values, sequential composition, function application, and pattern-matching. In addition, they include two kinds of algebraic operation calls [21, 22], ones the programmer can redefine using effect handlers [23], written $\mathsf{op}\ V\ (x\,.\,M)$, where op belongs to a fixed set $\mathscr{O}$ of operation symbols, and ones the programmer can use to introduce a time delay by blocking execution of the program for a set amount of time τ, written $\mathsf{delay}\ \tau\ M$. The latter are treated primitively in $\lambda_{[\tau]}$ and are not redefinable using effect handling $\mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N$. The last two computation terms (box and unbox) act as the introduction and elimination forms for the time-graded modal types used to specify temporal resources. In contrast to [1], where box is a value, in this paper we treat it as a computation, so as to give it an operational treatment following the tradition of FGCBV, where only computations execute and are given an operational semantics. The type system we review below then ensures that a box and corresponding unboxes interact only when enough time is spent between them.

Value	V,W	$::= x$	variable
		$\mid \mathsf{f}(V_1,\ldots,V_n)$	constant
		$\mid () \mid (V,W)$	unit and pairing
		$\mid \mathsf{fun}\ (x:X) \mapsto M$	function
Computation	M,N	$::= \mathsf{return}\ V$	returning a value
		$\mid \mathsf{let}\ x = M\ \mathsf{in}\ N$	sequential composition
		$\mid V\,W$	function application
		$\mid \mathsf{match}\ V\ \mathsf{with}\ \{(x,y) \mapsto N\}$	pattern-matching
		$\mid \mathsf{op}\ V\ (x\,.\,M)$	algebraic operation call
		$\mid \mathsf{delay}\ \tau\ M$	time delay operation
		$\mid \mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N$	effect handling
		$\mid \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N$	boxing up a temporal resource
		$\mid \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N$	unboxing a temporal resource
Effect handler	H	$::= (x\,.\,k\,.\,M_{\mathsf{op}})_{\mathsf{op}\in\mathscr{O}}$	operation clauses

Figure 1: Values, computations, and effect handlers of $\lambda_{[\tau]}$.

2.2 Types and Type System

Similarly to terms, $\lambda_{[\tau]}$ distinguishes between *value* and *computation types*, defined in Fig. 2. Value types include standard types from FGCBV: base, unit, product, and function types. In addition, they include the *time-graded modal type* $[\tau]X$, which is used to type and specify temporal resources. Its intuitive reading is: a value of type $[\tau]X$ denotes an X-typed resource that can be accessed and used after at least τ time units have passed, or differently, it gives a guarantee that in at most τ time units an X-typed resource is ready for use. For instance, in the code snippet given in Sect. 1, the variable partB" could be typed with the type $[\tau_{\text{primer-dry}}]\,\text{PartB}$ and partC"' could be typed with the type $[\tau_{\text{paint-dry}}]\,\text{PartC}$, where $\tau_{\text{primer-dry}}$ and

$\tau_{\text{paint-dry}}$ specify the amount of time that a part has to dry. The computation type $X \,!\, \tau$ specifies effectful computations that return X-typed values and spend τ time units doing that. As in FGCBV, the bodies of functions are effectful computations, so also the codomain of the function type gets a computation type.

For simplicity and concreteness, and similarly to [1], we only consider natural number valued grades τ for modal and computation types, and context modalities and operation signatures below, leaving a generalisation to other monoids of grades for future work. For instance, one option would be to replace the monoid of natural numbers with sequences of program events for all grades. Another option would be to consider different monoids, e.g., keeping natural numbers for the modal types and using sequences of events for computation types, connected by a monoid morphism that calculates a program run time from a sequence of events. This way one could change the granularity of type-based temporal specifications.

$$
\begin{array}{rl}
\text{Time grade:} & \tau \in \mathbb{N} \\
\text{Ground type } A, B, C & ::= \mathsf{b} \mid \mathsf{unit} \mid A \times B \mid [\tau]A \\
\text{Value type } X, Y, Z & ::= A \mid X \times Y \mid X \to Y \,!\, \tau \mid [\tau]X \\
\text{Computation type:} & X \,!\, \tau \\
\text{(Variable) context } \Gamma & ::= \emptyset \mid \Gamma, x{:}X \mid \Gamma, \langle \tau \rangle
\end{array}
$$

Figure 2: Types and contexts of $\lambda_{[\tau]}$.

Well-typed values and *computations* are specified using *typing judgements* $\Gamma \vdash V : X$ and $\Gamma \vdash M : X \,!\, \tau$. The *contexts* Γ are defined in Fig. 2. In addition to containing variables, they also contain *time-graded modalities* $\langle \tau \rangle$, which are used to give the time-graded modal type $[\tau]X$ a Fitch-style presentation [5, 7]. In addition, this modality is used to track time-passage in computations in the type system. For instance, when reading it from left to right, the context $\Gamma = x{:}X, \langle 4 \rangle, y{:}Y, z{:}Z, \langle 2 \rangle$ describes that first a variable x was brought into scope, then 4 time units passed, after which variables y and z were brought into scope, after which 2 more time units passed. This way one can keep track of how many time units ago temporal resources were brought into scope and whether they are allowed to be unboxed and used in a program (if enough time has passed since, as specified by a context). For readability, we omit the empty context $\emptyset$ whenever possible, e.g., when typing a closed value as $\vdash V : X$. The *composition* of two contexts Γ, Γ' is defined by recursion on Γ'. As usual, this composition is associative and the empty context $\emptyset$ is its *unit*.

For defining well-typed terms, it is assumed that every constant f is assigned a *base-typed signature* $\mathsf{f} : (\mathsf{b}_1, \ldots, \mathsf{b}_n) \to \mathsf{b}$ and that every algebraic operation op in $\mathscr{O}$ is assigned a *ground-typed time-graded signature* $\mathsf{op} : A_{\mathsf{op}} \to B_{\mathsf{op}} \,!\, \tau_{\mathsf{op}}$, where A_{op} is the type of inputs of op, B_{op} is the type of outputs of op, and τ_{op} specifies how much time executing op takes. Importantly, both A_{op} and B_{op} can include modal types.

The *typing rules* are given in Fig. 3. They are in large part standard from FGCBV, modulo the added context modalities and time-graded effect typing [11, 18, 25]. We only discuss the most important temporal features. Returning values (RETURN) does not take time, so it is typed with $X \,!\, 0$. In sequential composition (LET), the context of the continuation N is not only extended with the variable x denoting the value returned by M, but also with the modality $\langle \tau \rangle$, expressing that before x is bound and N starts executing an additional τ time units have passed. Similar temporal awareness also appears in the contexts of the continuations of algebraic operation calls (OP and DELAY). The major temporal addition to the typing of effect handlers and effect handling (HANDLE) involves wrapping the continuations k in modal

types to ensure that an operation's continuation is not resumed before the time prescribed by its signature.

$$\textsc{Var}\ \frac{x{:}X \in \Gamma}{\Gamma \vdash x : X} \qquad \textsc{Const}\ \frac{(\Gamma \vdash V_i : \mathsf{b}_i)_{1 \leq i \leq n}}{\Gamma \vdash \mathsf{f}(V_1, \ldots, V_n) : \mathsf{b}} \qquad \textsc{Pair}\ \frac{\Gamma \vdash V : X \qquad \Gamma \vdash W : Y}{\Gamma \vdash (V, W) : X \times Y} \qquad \textsc{Unit}\ \frac{}{\Gamma \vdash () : \mathsf{unit}}$$

$$\textsc{Fun}\ \frac{\Gamma, x{:}X \vdash M : Y \,!\, \tau}{\Gamma \vdash \mathsf{fun}\ (x : X) \mapsto M : X \to Y \,!\, \tau} \qquad \textsc{Return}\ \frac{\Gamma \vdash V : X}{\Gamma \vdash \mathsf{return}\ V : X \,!\, 0} \qquad \textsc{Let}\ \frac{\Gamma \vdash M : X \,!\, \tau \qquad \Gamma, \langle \tau \rangle, x{:}X \vdash N : Y \,!\, \tau'}{\Gamma \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : Y \,!\, \tau + \tau'}$$

$$\textsc{Apply}\ \frac{\Gamma \vdash V : X \to Y \,!\, \tau \qquad \Gamma \vdash W : X}{\Gamma \vdash V\, W : Y \,!\, \tau} \qquad \textsc{Match}\ \frac{\Gamma \vdash V : X \times Y \qquad \Gamma, x{:}X, y{:}Y \vdash N : Z \,!\, \tau}{\Gamma \vdash \mathsf{match}\ V\ \mathsf{with}\ \{(x, y) \mapsto N\} : Z \,!\, \tau}$$

$$\textsc{Op}\ \frac{\Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma, \langle \tau_{\mathsf{op}} \rangle, x{:}B_{\mathsf{op}} \vdash M : X \,!\, \tau}{\Gamma \vdash \mathsf{op}\ V\ (x\,.\,M) : X \,!\, \tau_{\mathsf{op}} + \tau} \qquad \textsc{Delay}\ \frac{\Gamma, \langle \tau \rangle \vdash M : X \,!\, \tau'}{\Gamma \vdash \mathsf{delay}\ \tau\, M : X \,!\, \tau + \tau'}$$

$$\textsc{Handle}\ \frac{\begin{array}{c}\Gamma \vdash M : X \,!\, \tau \qquad \Gamma, \langle \tau \rangle, z{:}X \vdash N : Y \,!\, \tau' \\ H = (x\,.\,k\,.\,M_{\mathsf{op}})_{\mathsf{op} \in \mathscr{O}} \qquad \left(\forall \tau''\,.\,\Gamma, x{:}A_{\mathsf{op}}, k{:}[\tau_{\mathsf{op}}]\,(B_{\mathsf{op}} \to Y \,!\, \tau'') \vdash M_{\mathsf{op}} : Y \,!\, \tau_{\mathsf{op}} + \tau''\right)_{\mathsf{op} \in \mathscr{O}}\end{array}}{\Gamma \vdash \mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N : Y \,!\, \tau + \tau'}$$

$$\textsc{Box}\ \frac{\Gamma, \langle \tau \rangle \vdash V : X \qquad \Gamma, x{:}[\tau]X \vdash N : Y \,!\, \tau'}{\Gamma \vdash \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N : Y \,!\, \tau'} \qquad \textsc{Unbox}\ \frac{\tau \leq \tau_\Gamma \qquad \Gamma - \tau \vdash V : [\tau]X \qquad \Gamma, x{:}X \vdash N : Y \,!\, \tau'}{\Gamma \vdash \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N : Y \,!\, \tau'}$$

Figure 3: Typing rules of $\lambda_{[\tau]}$.

The introduction rule for modal types (BOX) allows us to box up a value V at a modal type $[\tau]X$ if it is safe to use V at type X in the future after τ time units, as expressed by extending the context of V with the modality $\langle \tau \rangle$. Dually, the elimination rule for modal types (UNBOX) allows us to unbox a boxed value V, but only when enough time has passed since V was boxed up or brought into scope (e.g., returned by an algebraic operation call). This is ensured by typing V in a τ time units earlier context $\Gamma - \tau$. This operation, taking a context Γ to a τ time units earlier state, is defined recursively as follows:

$$\Gamma - 0 \stackrel{\text{def}}{=} \Gamma \qquad \emptyset - \tau_+ \stackrel{\text{def}}{=} \emptyset \qquad (\Gamma, x{:}X) - \tau_+ \stackrel{\text{def}}{=} \Gamma - \tau_+ \qquad (\Gamma, \langle \tau' \rangle) - \tau_+ \stackrel{\text{def}}{=} \begin{cases} \Gamma, \langle \tau' - \tau_+ \rangle, & \text{if } \tau_+ \leq \tau' \\ \Gamma - (\tau_+ - \tau'), & \text{otherwise} \end{cases}$$

where τ_+ denotes a non-zero natural number. For instance, if we take the context $x{:}X, \langle 4 \rangle, y{:}Y, z{:}Z, \langle 2 \rangle$ to a 5 time units earlier state, we are left with the context $x{:}X, \langle 1 \rangle$. Further, in the UNBOX rule, we write τ_Γ for the time passage encapsulated by the context Γ. This operation is defined recursively as follows:

$$\tau_\emptyset \stackrel{\text{def}}{=} 0 \qquad \tau_{(\Gamma, x{:}X)} \stackrel{\text{def}}{=} \tau_\Gamma \qquad \tau_{(\Gamma, \langle \tau \rangle)} \stackrel{\text{def}}{=} \tau_\Gamma + \tau$$

The condition $\tau \leq \tau_\Gamma$ then ensures that at least τ time units have passed according to Γ. It is useful to note that this condition, together with the operation $\Gamma - \tau$, makes the modal fragment of $\lambda_{[\tau]}$ an instance of modal λ-calculi based on parametric right adjoints [7]. See [1] for more discussion on this connection.

In the rest of the paper, we find it convenient to write $\Gamma_{x,1}, x{:}X, \Gamma_{x,2} = \Gamma$ for the *splitting* of Γ by some variable $x{:}X$ in it. In particular, using this notation, we have for every $x : X \in \Gamma - \tau$, that $\tau \leq \tau_{\Gamma_{x,2}}$.

2.3 Renamings and Admissible Rules

The value and computation typing judgements are closed under a number of expected admissible rules.

Proposition 2.1. *Standard structural rules are admissible (where J ranges over both judgements):*

$$\frac{\Gamma,\Gamma' \vdash J \qquad x{:}X \notin \Gamma,\Gamma'}{\Gamma,x{:}X,\Gamma' \vdash J} \qquad \frac{\Gamma,x{:}X,y{:}Y,\Gamma' \vdash J}{\Gamma,y{:}Y,x{:}X,\Gamma' \vdash J} \qquad \frac{\Gamma,x{:}X,x'{:}X,\Gamma' \vdash J}{\Gamma,x{:}X,\Gamma' \vdash J[x/x']}$$

and additionally, the following typing rules are admissible for the time-graded context modalities:

$$\frac{\Gamma,\langle 0\rangle,\Gamma' \vdash J}{\Gamma,\Gamma' \vdash J} \qquad \frac{\Gamma,\langle \tau_1+\tau_2\rangle,\Gamma' \vdash J}{\Gamma,\langle \tau_1\rangle,\langle \tau_2\rangle,\Gamma' \vdash J} \qquad \frac{\Gamma,\langle \tau\rangle,\Gamma' \vdash J \quad \tau \leq \tau'}{\Gamma,\langle \tau'\rangle,\Gamma' \vdash J} \qquad \frac{\Gamma,\langle \tau\rangle,x{:}X,\Gamma' \vdash J}{\Gamma,x{:}X,\langle \tau\rangle,\Gamma' \vdash J}$$

As in [1], this is easiest to prove by defining an appropriate notion of variable renamings, with the desired rules then following from the action of specific renamings on well-typed values and computations (using Prop. 2.2 stated below). Compared to [1], where such variable renamings were defined simply as an inductively defined set, here we give them an equivalent, but more computational definition, directly as certain functions from variables in one context to variables in another.

We write $\mathsf{Ren}\,\Gamma\,\Gamma'$ for the set of *type-preserving renamings* from the variables in context Γ to variables in Γ'. For a $\rho \in \mathsf{Ren}\,\Gamma\,\Gamma'$, given an $x{:}X$ in Γ, we write $\rho(x){:}X$ for the corresponding variable in Γ'. In order to ensure that renamings preserve all the desired temporal properties, we require additionally that $\tau_\Gamma \leq \tau_{\Gamma'}$, and furthermore, that $\tau_{\Gamma_{x,2}} \leq \tau_{\Gamma'_{\rho(x),2}}$, for all $x{:}X \in \Gamma$. That is, the set $\mathsf{Ren}\,\Gamma\,\Gamma'$ is defined as

$$\mathsf{Ren}\,\Gamma\,\Gamma' \stackrel{\text{def}}{=} \left\{\rho : \mathit{vars}(\Gamma) \to \mathit{vars}(\Gamma') \;\middle|\; \tau_\Gamma \leq \tau_{\Gamma'} \wedge \forall(x{:}X \in \Gamma).\, \mathit{type}(\rho(x)) = X \wedge \tau_{\Gamma_{x,2}} \leq \tau_{\Gamma'_{\rho(x),2}}\right\}$$

Our contribution is showing that in this way we can recover renamings corresponding to all the inductive cases used in [1], such as, $\mathsf{var}^{\mathrm{r}}_{x:X\in\Gamma} \in \mathsf{Ren}\,(\Gamma,y{:}X)\,\Gamma$ and $\mu^{\mathrm{r}} \in \mathsf{Ren}\,(\Gamma,\langle\tau_1+\tau_2\rangle)\,(\Gamma,\langle\tau_1\rangle,\langle\tau_2\rangle)$. The other direction of this equivalence, namely, defining a function in $\mathsf{Ren}\,\Gamma\,\Gamma'$ from an inductively defined renaming, was already present in [1]. This way we can also define many other renamings useful later on, such as different flavours of weakening renamings $\mathsf{wk}_{\Gamma'} \in \mathsf{Ren}\,\Gamma\,(\Gamma,\Gamma')$ and $\mathsf{wk}_{\tau\leq\tau_{\Gamma'}} \in \mathsf{Ren}\,(\Gamma,\langle\tau\rangle)\,(\Gamma,\Gamma')$, when $\tau \leq \tau_{\Gamma'}$, and a general congruence renaming $\mathsf{cong}_{\Gamma''}(\rho) \in \mathsf{Ren}\,(\Gamma,\Gamma'')\,(\Gamma',\Gamma'')$, where $\rho \in \mathsf{Ren}\,\Gamma\,\Gamma'$.

Given a renaming $\rho \in \mathsf{Ren}\,\Gamma\,\Gamma'$, the *action of ρ on values and computations*, written $V[\rho]$ and $M[\rho]$, respectively, is defined by straightforward mutual recursion on the structure of V and M. For renamings ρ that do not change variable names or perform substitutions (such as weakening a context with new variables or time passage), we typically omit ρ in $V[\rho]$ and $M[\rho]$, and simply write V and M.

Proposition 2.2. *If $\rho \in \mathsf{Ren}\,\Gamma\,\Gamma'$ and $\Gamma \vdash V : X$ and $\Gamma \vdash M : X\,!\,\tau$, then $\Gamma' \vdash V[\rho] : X$ and $\Gamma' \vdash M[\rho] : X\,!\,\tau$.*

We conclude the overview of $\lambda_{[\tau]}$ by noting that the standard rules for substitution are also admissible.

Proposition 2.3. *If $\Gamma,x{:}X,\Gamma' \vdash J$, $\Gamma \vdash W : X$, then $\Gamma,\Gamma' \vdash J[W/x]$, where J ranges over both judgements.*

3 Stateful Time-Aware Operational Semantics

In this section we equip $\lambda_{[\tau]}$ with an operational semantics. As the purpose of $\lambda_{[\tau]}$ is to provide a core calculus for safe programming with temporal resources, where modal types are used to control when such resources can be used and a time-graded effect system is used to track the run times of computations, we wish the operational semantics to reflect these aspects of $\lambda_{[\tau]}$, and to provide an alternative, dynamic means for analysing the use of temporal resources and computations' run times. To this end, we equip $\lambda_{[\tau]}$ with a stateful time-aware operational semantics, in which the states both track which resources are being used by a program and provide a cost model for $\lambda_{[\tau]}$-programs by tracking their run time.

3.1 States

We begin by defining the *states* $\mathbb{S}$ that we use in our operational semantics. They are given by

$$\mathbb{S} ::= \emptyset \mid \mathbb{S}, \langle \tau \rangle \mid \mathbb{S}, x \mapsto_{[\tau]X} V$$

A state is either empty (denoting that no resources have been boxed up and no time has passed in a computation), an extension of a prior state with the time-graded modality $\langle \tau \rangle$ (denoting that extra τ units of time have passed), or an extension of a prior state with the resource mapping $x \mapsto_{[\tau]X} V$ (denoting that the value V has been boxed up in a computation). Analogously to contexts Γ, we also define operations $\mathbb{S} - \tau$ and $\tau_{\mathbb{S}}$ on states. Their definitions are analogous to the operations on contexts, so we omit them.

In order to later be able to talk about well-typed computations being reduced in a state, we recursively translate each state $\mathbb{S}$ into a corresponding context $\Gamma_{\mathbb{S}}$, given by the following definition:

$$\Gamma_{\mathbb{S}} \stackrel{\text{def}}{=} \begin{cases} \emptyset, & \text{if } \mathbb{S} = \emptyset \\ \Gamma_{\mathbb{S}'}, \langle \tau \rangle, & \text{if } \mathbb{S} = \mathbb{S}', \langle \tau \rangle \\ \Gamma_{\mathbb{S}'}, x : [\tau] X, & \text{if } \mathbb{S} = \mathbb{S}', x \mapsto_{[\tau]X} V \end{cases}$$

We also define the *composition* of two states $\mathbb{S}, \mathbb{S}'$, by straightforward structural recursion on $\mathbb{S}'$. It is associative, has the empty state $\emptyset$ as its *unit*, and it is related to operations on contexts as follows:

Proposition 3.1.

1. *For all* $\mathbb{S}$ *and* τ*, we have* $\Gamma_{\mathbb{S}-\tau} = \Gamma_{\mathbb{S}} - \tau$.
2. *For all* $\mathbb{S}$ *and* $\mathbb{S}'$*, we have* $\Gamma_{\mathbb{S},\mathbb{S}'} = \Gamma_{\mathbb{S}}, \Gamma_{\mathbb{S}'}$.
3. *For all* $\mathbb{S}$*, we have* $\tau_{\Gamma_{\mathbb{S}}} = \tau_{\mathbb{S}}$.
4. *For all* $\mathbb{S}$ *and* $\mathbb{S}'$*, we have* $\tau_{\mathbb{S},\mathbb{S}'} = \tau_{\mathbb{S}} + \tau_{\mathbb{S}'}$.

It is also useful to formally note that variables in the context $\Gamma_{\mathbb{S}}$ get assigned very specific types.

Proposition 3.2. *If* $x : X \in \Gamma_{\mathbb{S}}$*, then* $X = [\tau]Y$ *for some* τ *and* Y*, and also,* $x \mapsto_{[\tau]Y} V \in \mathbb{S}$*, for some* V.

Next, in order to express that in our operational semantics the states can only ever monotonically "grow", by extensions with time or resources, we define the following partial order $\mathbb{S} \leq \mathbb{S}'$ on states:

$$\frac{}{\mathbb{S} \leq \mathbb{S}} \qquad \frac{\mathbb{S} \leq \mathbb{S}'}{\mathbb{S} \leq (\mathbb{S}', \langle \tau \rangle)} \qquad \frac{\mathbb{S} \leq \mathbb{S}'}{\mathbb{S} \leq (\mathbb{S}', x \mapsto_{[\tau]X} V)}$$

The monotonic growth aspect of states manifests both as state splittings and weakening renamings.

Proposition 3.3.

1. *If* $\mathbb{S} \leq \mathbb{S}'$*, then* $\mathbb{S}' = \mathbb{S}, \mathbb{S}''$*, for some* $\mathbb{S}''$.
2. *If* $\mathbb{S} \leq \mathbb{S}'$*, then* $\tau_{\mathbb{S}} \leq \tau_{\mathbb{S}'}$.
3. *If* $\mathbb{S} \leq \mathbb{S}'$*, then there is a corresponding weakening renaming* $\mathsf{wk}_{\mathbb{S} \leq \mathbb{S}'} : \Gamma_{\mathbb{S}} \rightsquigarrow \Gamma_{\mathbb{S}'}$.

Finally, for proving the correctness of our semantics, we identify *well-formed* states $\Gamma \vdash \mathbb{S}$, given by

$$\frac{}{\Gamma \vdash \emptyset} \qquad \frac{\Gamma \vdash \mathbb{S}}{\Gamma \vdash \mathbb{S}, \langle \tau \rangle} \qquad \frac{\Gamma \vdash \mathbb{S} \qquad \Gamma, \Gamma_{\mathbb{S}}, \langle \tau \rangle \vdash V : X \qquad x \notin \Gamma, \Gamma_{\mathbb{S}}}{\Gamma \vdash \mathbb{S}, x \mapsto_{[\tau]X} V}$$

We define the relation $\Gamma \vdash \mathbb{S}$ in a potentially open context Γ because that allows us to talk about the well-formedness of composite states, namely, if $\Gamma \vdash \mathbb{S}$ and $\Gamma, \Gamma_{\mathbb{S}} \vdash \mathbb{S}'$, then we can show that $\Gamma \vdash \mathbb{S}, \mathbb{S}'$.

3.2 Small-Step Reduction Relation

At the core of our operational semantics is the *small-step reduction relation* $\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle$ that describes how a computation M can make one reduction step to a computation M', with the step starting in an initial state $\mathbb{S}$ and ending in a final state $\mathbb{S}'$. The rules defining this relation are given in Fig. 4.

SEM-APP

$$\langle \mathbb{S} \mid (\mathsf{fun}\ (x{:}X) \mapsto M)\, V \rangle \rightsquigarrow \langle \mathbb{S} \mid M[V/x] \rangle$$

SEM-MATCH

$$\langle \mathbb{S} \mid \mathsf{match}\ (V,W)\ \mathsf{with}\ \{(x,y) \mapsto N\} \rangle \rightsquigarrow \langle \mathbb{S} \mid N[V/x, W/y] \rangle$$

SEM-LET-CONG

$$\frac{\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle}{\langle \mathbb{S} \mid \mathsf{let}\ x = M\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S}' \mid \mathsf{let}\ x = M'\ \mathsf{in}\ N \rangle}$$

SEM-LET-RET

$$\langle \mathbb{S} \mid \mathsf{let}\ x = (\mathsf{return}\ V)\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S} \mid N[V/x] \rangle$$

SEM-LET-OP

$$\langle \mathbb{S} \mid \mathsf{let}\ x = (\mathsf{op}\ V\ (y\ .\ M))\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S} \mid \mathsf{op}\ V\ (y\ .\ \mathsf{let}\ x = M\ \mathsf{in}\ N) \rangle$$

SEM-HANDLE-CONG

$$\frac{\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle}{\langle \mathbb{S} \mid \mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S}' \mid \mathsf{handle}\ M'\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \rangle}$$

SEM-HANDLE-RET

$$\langle \mathbb{S} \mid \mathsf{handle}\ (\mathsf{return}\ V)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S} \mid N[V/z] \rangle$$

SEM-HANDLE-OP

$$\frac{H = (x\ .\ k\ .\ M_{\mathsf{op}})_{\mathsf{op} \in \mathscr{O}}}{\langle \mathbb{S} \mid \mathsf{handle}\ (\mathsf{op}\ V\ (y\ .\ M))\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S} \mid \mathsf{box}\ (\mathsf{fun}\ (y : B_{\mathsf{op}}) \mapsto \mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N)\ \mathsf{as}\ w\ \mathsf{in}\ M_{\mathsf{op}}[V/x, w/k] \rangle}$$

SEM-DELAY

$$\langle \mathbb{S} \mid \mathsf{delay}\ \tau\ M \rangle \rightsquigarrow \langle \mathbb{S}, \langle \tau \rangle \mid M \rangle$$

SEM-BOX

$$\langle \mathbb{S} \mid \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S}, x \mapsto_{[\tau]X} V \mid N \rangle$$

SEM-UNBOX

$$\frac{y \in \mathbb{S}}{\langle \mathbb{S} \mid \mathsf{unbox}_{[\tau]X}\ y\ \mathsf{as}\ x\ \mathsf{in}\ N \rangle \rightsquigarrow \langle \mathbb{S} \mid N[\mathbb{S}[y]/x] \rangle}$$

Figure 4: Stateful time-aware operational semantics for $\lambda_{[\tau]}$.

The rules corresponding to β-reduction steps for the computations inherited from FGCBV are standard, these are SEM-APP, SEM-MATCH, and SEM-LET-RET. In particular, they do not change the state.

The rule SEM-LET-OP allows operations to be pulled out from sequential composition. The rules SEM-LET-CONG and SEM-HANDLE-CONG allow the first computation in sequential composition and the computation being handled to make progress. The rule SEM-HANDLE-RET describes how return values are handled. The rule SEM-HANDLE-OP expresses how effect handling replaces an algebraic operation call with the corresponding case of the effect handler. This rule differs from the standard operation handling rule [10, 23] in that here we have to box up the recursive call to effect handling as a temporal resource before substituting it for the continuation k, as required by the typing of k.

The rule SEM-DELAY models the blocking of execution for a prescribed amount of time by delay by extending the state by that time amount, with nothing else useful happening during that time period.

The rules SEM-BOX and SEM-UNBOX express how box and unbox behave analogously to memory allocation and lookup operations. On the one hand, when the next computation to be executed is box,

then the state is extended with the resource being boxed up and the continuation is executed next. On the other hand, when the next computation to be executed is unbox, then we look up the corresponding value (that was boxed up earlier) in the state and substitute it into the continuation before executing it. Observe that this rule has a side-condition requiring the resource y being unboxed to be in the state $\mathbb{S}$, ensuring that the following *a priori* partial *lookup operation* is defined in the right-hand side of the rule:

$$\mathbb{S}[x] \stackrel{\text{def}}{=} \begin{cases} V, & \text{if } \mathbb{S} = \mathbb{S}', x \mapsto_{[\tau]X} V \\ \mathbb{S}'[x], & \text{if } \mathbb{S} = \mathbb{S}', \langle \tau \rangle \text{ or } \mathbb{S} = \mathbb{S}', y \mapsto_{[\tau]X} V \text{ and } x \neq y \\ \text{undefined}, & \text{if } \mathbb{S} = \emptyset \end{cases}$$

3.3 Type-Safety

We now prove type-safety for the semantics we defined in the previous section. As standard, we split type-safety into theorems of *progress* (Thm. 3.7) and *type preservation* (Thm. 3.10) [29]. However, before proving those theorems, we first establish temporal variants of commonly proved useful lemmas.

First, we formalise the intuition that states can only monotonically grow during program execution.

Proposition 3.4. *If* $\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle$, *then* $\mathbb{S} \leq \mathbb{S}'$.

Next, we have a standard canonicity result for values. While this result is commonly proved for values in an empty context, we have to prove it in open contexts of the form $\Gamma_{\mathbb{S}}$ because both values and computations can contain variables referring to the resources stored in the state during execution.

Proposition 3.5. *If* $\Gamma_{\mathbb{S}} \vdash V : X \times Y$, *then there exist* W_1 *and* W_2, *such that* $V = (W_1, W_2)$, *with* $\Gamma_{\mathbb{S}} \vdash W_1 : X$ *and* $\Gamma_{\mathbb{S}} \vdash W_2 : Y$. *Well-typed values in* $\Gamma_{\mathbb{S}}$ *of unit and function type are also in their introduction forms.*

The canonicity for values of the modal type $[\tau]X$ is a bit different. Namely, as there is no value introduction form for $[\tau]X$, then values of this type in a context $\Gamma_{\mathbb{S}}$ are necessarily variables in $\Gamma_{\mathbb{S}}$.

Proposition 3.6. *If* $\Gamma_{\mathbb{S}} \vdash V : [\tau]X$, *then* $V = y$ *for some* $y : [\tau]X \in \Gamma_{\mathbb{S}}$, *or for* $y \mapsto_{[\tau]X} W \in \mathbb{S}$, *for some* W.

For stating and proving the progress theorem, we say that a computation M is in a *result form*, if either $M = \mathsf{return}\ V$, for some value V, or $M = \mathsf{op}\ W\ (x\ .\ N)$, for some value W and computation N. These result forms are standard from calculi for algebraic effects and effect handlers [10].

The progress theorem then takes the following shape. While it is commonly stated in an empty context, then analogously to the canonicity lemmas above, we have to state it in contexts of the form $\Gamma_{\mathbb{S}}$.

Theorem 3.7 (Progress)**.** *If* $\vdash \mathbb{S}$ *and* $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$, *then either* M *is in a result form and has stopped reducing, or it is possible to make a reduction step* $\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle$, *for some* $\mathbb{S}'$ *and* M'.

Proof. The proof proceeds as usual by induction on the derivation of $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$. See the Agda code or the second author's MSc thesis [30] for details. Here we only mention that in order to be able to apply the rule SEM-UNBOX, the side-condition $y \in \mathbb{S}$ is satisfied because by applying Prop. 3.6 to the typing derivation of $\Gamma_{\mathbb{S}} - \tau \vdash V : [\tau]X$, we get that $V = y$ and $y : [\tau]X \in \Gamma_{\mathbb{S}} - \tau$. By Prop. 3.1, this is the same as $y : [\tau]X \in \Gamma_{\mathbb{S}-\tau}$, which is the same as $y \in \mathbb{S} - \tau$ according to Prop. 3.2. As a result, $y \in \mathbb{S}$, as required. □

For proving the progress theorem, we first establish standard inversion lemmas for typing rules.

Proposition 3.8. *If* $\Gamma \vdash (V_1, V_2) : X$, *then* $X = Y_1 \times Y_2$ *and* $\Gamma \vdash V_i : Y_i$, *and analogously for other terms.*

As there are no value introduction forms for $[\tau]X$, then for its inversion-like lemma we establish the following result, showing that values of type $[\tau]X$ in $\Gamma_{\mathbb{S}}$ correspond to well-typed underlying X-typed values. As we established in Prop. 3.6 that only such values are variables, we state this result as follows:

Proposition 3.9. *If* $\vdash \mathbb{S}$ *and* $x:[\tau]X \in \Gamma_{\mathbb{S}}$*, then* $(\Gamma_{\mathbb{S}})_{x,1}, \langle\tau\rangle \vdash \mathbb{S}[x] : X$.

We now prove the preservation theorem. It differs slightly from its usual form because applying the reduction rules of our semantics can result in a computation whose run time is shorter than the original one's, e.g., when the original computation is a delay. However, the time lost from the computation's run time is accumulated in the state, giving us an invariant that expresses temporal type preservation.

Theorem 3.10 (Preservation)**.** *If* $\vdash \mathbb{S}$ *and* $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$*, and if* $\langle \mathbb{S} \mid M\rangle \rightsquigarrow \langle \mathbb{S}' \mid M'\rangle$*, for some* $\mathbb{S}'$ *and* M'*, then* $\vdash \mathbb{S}'$ *and there exists a* τ'*, such that* $\tau_{\mathbb{S}} + \tau = \tau_{\mathbb{S}'} + \tau'$ *and* $\Gamma_{\mathbb{S}'} \vdash M' : X \,!\, \tau'$.

Proof. The proof proceeds as usual by induction on the derivation of $\langle \mathbb{S} \mid M\rangle \rightsquigarrow \langle \mathbb{S}' \mid M'\rangle$, using Prop. 3.8 to invert the typing of $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$ based on the structure forced upon M by the reduction rules. See the Agda code or the second author's MSc thesis [30] for details. Here we only note that in the case of SEM-UNBOX, where $M = \mathsf{unbox}_{[\tau'']Y}\ y\ \mathsf{as}\ x\ \mathsf{in}\ N$, inverting the typing of $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$ and applying Prop. 3.6 gives us that $y:[\tau'']Y \in \Gamma_{\mathbb{S}} - \tau''$, which is the same as $y:[\tau'']Y \in \Gamma_{\mathbb{S}-\tau''}$ by Prop. 3.1. Next, applying Prop. 3.9, we get that $(\Gamma_{\mathbb{S}-\tau''})_{y,1}, \langle\tau''\rangle \vdash (\mathbb{S} - \tau'')[y] : Y$, which is the same as $(\Gamma_{\mathbb{S}-\tau''})_{y,1}, \langle\tau''\rangle \vdash \mathbb{S}[y] : Y$, because $\vdash \mathbb{S}$ contains no repeated variables. Finally, by observing that $(\Gamma_{\mathbb{S}-\tau''})_{y,1} = (\Gamma_{\mathbb{S}})_{y,1}$ and $\tau_{(\Gamma_{\mathbb{S}-\tau''})_{y,2}} + \tau'' = \tau_{(\Gamma_{\mathbb{S}})_{y,2}}$, which means that $\tau'' \leq \tau_{(\Gamma_{\mathbb{S}})_{y,2}} = \tau_{y:[\tau'']Y,(\Gamma_{\mathbb{S}})_{y,2}}$, we can apply the renaming $\mathsf{wk}_{\tau'' \leq \tau_{y:[\tau'']Y,(\Gamma_{\mathbb{S}})_{y,2}}} \in \mathsf{Ren}\ ((\Gamma_{\mathbb{S}})_{y,1}, \langle\tau''\rangle)\ \Gamma_{\mathbb{S}}$ to show that $\Gamma_{\mathbb{S}} \vdash \mathbb{S}[y] : Y$, giving us $\Gamma_{\mathbb{S}} \vdash N[\mathbb{S}[y]/x] : X \,!\, \tau$. □

3.4 Modelling Top-Level Runtime Environment

We conclude this section by noting that in our operational semantics any unhandled algebraic operation calls will result in the evaluation stopping in a result form. In order to keep the presentation simpler and to the point, we did not include any support for default "top-level" implementations of unhandled algebraic operation calls, in particular, as there are many different ways how this could be achieved. In an implementation based on $\lambda_{[\tau]}$, we envisage the operational semantics defined in this paper being intertwined with a top-level runtime, which executes unhandled operation calls and then hands the control back to the operational semantics. These implementations of algebraic operations could be as simple as a fixed rule $\langle \mathbb{S} \mid \mathsf{op}\ V\ (x\,.\,M)\rangle \rightsquigarrow \langle \mathbb{S}, \langle\tau_{\mathsf{op}}\rangle \mid M[\mathsf{val}_{\mathsf{op}}(V)/x]\rangle$ for each $\mathsf{op} : A_{\mathsf{op}} \to B_{\mathsf{op}} \,!\, \tau_{\mathsf{op}}$, where $\mathsf{val}_{\mathsf{op}}(V)$ is some value of type B_{op} depending on V. However, one can imagine that these implementations could be user-provided as well, either as an effect handler or a runner [2]. Moreover, such implementations could also be given access to the runtime of the implementation language, such as to OCaml or Haskell. We leave exploring all these different options for future work on a prototype implementation of $\lambda_{[\tau]}$.

4 Equational Soundness

In this section we accompany the basic type-safety results of the previous section with an additional correctness guarantee, expressed as a soundness theorem with respect to a temporal variant of the standard equational presentation of the computational effect of local state [19, 21, 26]. This way we are able to demonstrate that our stateful operational semantics agrees with a well-understood notion of computation.

4.1 Computation Term Contexts

Before we can define the equational theory of interest, we first define *computation term contexts* $\mathbb{K}$, which we use in the equational theory to abstract over computations that have to take enough time to run:

$$\mathbb{K} ::= [\,] \mid \mathsf{op}\ V\ (x\,.\,\mathbb{K}) \mid \mathsf{delay}\ \tau\ \mathbb{K} \mid \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K} \mid \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K}$$

We write the *composition* of two computation term contexts $\mathbb{K}$ and $\mathbb{K}'$ as $\mathbb{K}[\mathbb{K}']$, defined recursively by filling the hole in $\mathbb{K}$ with $\mathbb{K}'$. This composition is associative and the hole $[\,]$ is its *unit*. By a similarly straightforward recursion, we also compute *how much time* is spent in a $\mathbb{K}$ (by calling algebraic operations and delays) before arriving at the hole $[\,]$, and *which variables are bound* between the root of $\mathbb{K}$ and the hole $[\,]$, which we write as $\tau_{\mathbb{K}}$ and $\Gamma_{\mathbb{K}}$, respectively. The latter includes all the variables bound between the root of K and $[\,]$, both those bound by algebraic operations, and those bound by box and unbox.

Well-formed computation term contexts are defined using the judgement $\Gamma \vdash \mathbb{K} : \tau$, given by the rules

$$\frac{}{\Gamma \vdash [\,] : 0} \qquad \frac{\Gamma \vdash V : A_{\mathsf{op}} \qquad \Gamma, \langle \tau_{\mathsf{op}} \rangle, x : B_{\mathsf{op}} \vdash \mathbb{K} : \tau}{\Gamma \vdash \mathsf{op}\ V\ (x\,.\,\mathbb{K}) : \tau_{\mathsf{op}} + \tau} \qquad \frac{\Gamma, \langle \tau \rangle \vdash \mathbb{K} : \tau'}{\Gamma \vdash \mathsf{delay}\ \tau\ \mathbb{K} : \tau + \tau'}$$

$$\frac{\Gamma, \langle \tau \rangle \vdash V : X \qquad \Gamma, x : [\tau] X \vdash \mathbb{K} : \tau'}{\Gamma \vdash \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K} : \tau'} \qquad \frac{\tau \leq \tau_\Gamma \qquad \Gamma - \tau \vdash V : [\tau] X \qquad \Gamma, x : X \vdash \mathbb{K} : \tau'}{\Gamma \vdash \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K} : \tau'}$$

Observe that the type of a well-formed computation term context $\Gamma \vdash \mathbb{K} : \tau$ specifies only its run time τ, which is easy to see to coincide with the time $\tau_{\mathbb{K}}$ computed above. The judgement is polymorphic in the type of return values, allowing us to *fill the hole* in a $\mathbb{K}$ with any computation term M, written as $\mathbb{K}[M]$.

Both $\mathbb{K}[\mathbb{K}']$ and $\mathbb{K}[M]$ preserve the well-typedness of the involved computation term contexts and computations, as shown in the next proposition. In particular, observe that $\mathbb{K}'$ and M can refer to the variables $\Gamma_{\mathbb{K}}$ bound by $\mathbb{K}$, and that the run times are added together, analogously to let and handle.

Proposition 4.1.

1. *If* $\Gamma \vdash \mathbb{K} : \tau$ *and* $\Gamma, \Gamma_{\mathbb{K}} \vdash \mathbb{K}' : \tau'$, *then* $\Gamma \vdash \mathbb{K}[\mathbb{K}'] : \tau + \tau'$.
2. *If* $\Gamma \vdash \mathbb{K} : \tau$ *and* $\Gamma, \Gamma_{\mathbb{K}} \vdash M : X\,!\,\tau'$, *then* $\Gamma \vdash \mathbb{K}[M] : X\,!\,\tau + \tau'$.

Further, by straightforward recursion, we define actions of renamings and substitution on computation term contexts, written $\mathbb{K}[\rho]$ and $\mathbb{K}[V/x]$, respectively. They satisfy analogous properties to Props. 2.2 and 2.3. Additionally, they are related to the respective operations on computations as follows:

Proposition 4.2.

1. *If* $\rho : \mathsf{Ren}\ \Gamma\ \Gamma'$ *and* $\Gamma \vdash \mathbb{K} : \tau$ *and* $\Gamma, \Gamma_{\mathbb{K}} \vdash M : X\,!\,\tau'$, *then* $\mathbb{K}[M][\rho] = \mathbb{K}[\rho][M[\mathsf{cong}_{\Gamma_{\mathbb{K}}}(\rho)]]$.
2. *If* $\Gamma, x : X, \Gamma' \vdash \mathbb{K} : \tau$ *and* $\Gamma, x : X, \Gamma', \Gamma_{\mathbb{K}} \vdash M : Y\,!\,\tau'$ *and* $\Gamma \vdash V : X$, *then* $\mathbb{K}[M][V/x] = \mathbb{K}[V/x][M[V/x]]$.

As before, we omit renamings ρ when they do not change variable names or perform substitution.

4.2 Equational Theory for Temporal Resources

We define the equational theory using two mutually inductively defined *equality judgements*

$$\Gamma \vdash V \equiv W : X \qquad\qquad \Gamma \vdash M \equiv N : X\,!\,\tau$$

where we presuppose that we only compare well-typed values and computations. For most part the contexts and types play no significant role in equations, so we omit them wherever convenient.

These judgements are defined using rules corresponding to different kinds of equations. We postpone the full list of rules to Appendix A and in the paper discuss the temporally more important ones.

First, we include rules making these judgements into congruences. To these we add standard β- and η-equations for the non-modal $\lambda_{[\tau]}$-values and -computations, as in FGCBV [16, 17] e.g., for fun we add

$$\begin{aligned} (\mathsf{fun}\ (x:X) \mapsto M)\, V &\equiv M[V/x] && (\beta\text{-equation}) \\ V &\equiv \mathsf{fun}\ (x:X) \mapsto V\, x && (\eta\text{-equation, where } \Gamma \vdash V : X \to Y\,!\,\tau) \end{aligned}$$

Algebraicity

$$\mathsf{let}\ x = (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{in}\ N \equiv \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)$$
$$\mathsf{let}\ x = (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{in}\ N \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)$$
$$\mathsf{handle}\ (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N)$$
$$\mathsf{handle}\ (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N)$$

(with $y \notin fv(N)$ in all four equations)

Commutativity

$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{box}_{[\tau']Y}\ W\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{box}_{[\tau']Y}\ W\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N)$$
$$\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N)$$
$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N)$$

(with $x \notin fv(W), y \notin fv(V)$ in all three equations)

Interaction

$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K}[\mathsf{unbox}_{[\tau]X}\ x\ \mathsf{as}\ y\ \mathsf{in}\ N] \equiv \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K}[N[V/y]] \qquad (\tau_{\mathbb{K}} \geq \tau)$$

Discarding and Contraction

$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N \equiv N \qquad (x \notin fv(N))$$
$$\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N \equiv N \qquad (x \notin fv(N))$$
$$\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N[x/y]$$

Figure 5: box and unbox fragment of the equational theory of $\lambda_{[\tau]}$.

We also add standard equations for computation terms that we inherit from FGCBV, e.g., for let we add

$$\mathsf{let}\ x = (\mathsf{return}\ V)\ \mathsf{in}\ N \equiv N[V/x] \qquad (\beta\text{-equation for let})$$
$$\mathsf{let}\ x = (\mathsf{op}\ V\ (y\ .\ M))\ \mathsf{in}\ N \equiv \mathsf{op}\ V\ (y\ .\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)) \qquad (\text{algebraicity equation for let})$$

To those we add equations for effect handling, which are analogous to SEM-HANDLE-RET and SEM-HANDLE-OP from Fig. 4. We also include equations describing how delay interacts with 0 and $+$, and that it can be pulled out from sequential composition and effect handling, analogously to SEM-LET-OP.

Finally, we add equations pertinent to the behaviour of box and unbox. They are listed in Fig. 5. As foreshadowed earlier, they are very closely related to standard equational presentations of local state [19, 21, 26], by intuitively understanding box as memory allocation and unbox as a lookup operation, and by understanding the modal type $[\tau]X$ as the type of memory locations or reference cells storing X-typed values. Differently from the standard presentation of local state, we do not have an assignment operation.

The *algebraicity equations* allow us to pull box and unbox out of sequential composition, and they express that user-defined effect handlers do not handle these two constructs. The *commutativity equations* allow us to exchange the order of boxes and unboxes as long as no variables escape their scope.

The *interaction equation* is the main port of departure of our equational theory from the usual equational presentations of local state. While in the theory of local state the allocation and lookup operations interact when sequenced immediately after one another, for $\lambda_{[\tau]}$ it is crucial that we abstract over inter-

mediate computations with the computation term contexts $\mathbb{K}$, so as to ensure that enough time (at least τ) is spent between a box and a corresponding unbox, as enforced by the side-condition $\tau_{\mathbb{K}} \geq \tau$.

The two *discarding* equations remove box and unbox when they no longer influence the computation. The *contraction* equation expresses that multiple unboxings of the same resource yield the same value.

We can then prove general congruence and algebraicity results for computation term contexts.

Proposition 4.3. *If* $\Gamma \vdash \mathbb{K} : \tau$ *and* $\Gamma, \Gamma_{\mathbb{K}} \vdash M \equiv N : X \,!\, \tau'$*, then we have* $\Gamma \vdash \mathbb{K}[M] \equiv \mathbb{K}[N] : X \,!\, \tau + \tau'$.

Proposition 4.4. *If* $\Gamma \vdash \mathbb{K} : \tau$ *and* $\Gamma, \Gamma_{\mathbb{K}} \vdash M : X \,!\, \tau'$ *and* $\Gamma, \langle \tau + \tau' \rangle, x{:}X \vdash N : Y \,!\, \tau''$*, then we have the algebraicity equation* $\Gamma \vdash \mathsf{let}\ x = \mathbb{K}[M]\ \mathsf{in}\ N \equiv \mathbb{K}[\mathsf{let}\ x = M\ \mathsf{in}\ N] : Y \,!\, \tau + \tau' + \tau''$.

The equational theory defined in this paper differs from the one in the original paper [1] by box and unbox being treated here in an effectful, stateful way. In [1], box was considered a value, and the corresponding β- and η-equations described unbox simply as a pattern-matching eliminator for box.

4.3 Equational Soundness for Temporal Resources

We now prove that our operational semantics is sound with respect to the equational theory we defined.

4.3.1 From States to Computation Term Contexts

We begin by translating states $\mathbb{S}$ to computation term contexts $\mathbb{K}_{\mathbb{S}}$, so as to surround the computations appearing in our operational semantics with boxes corresponding to resources appearing in $\mathbb{S}$, thus allowing us to use the interaction equation on the unboxes in the computations. The computation term context $\mathbb{K}_{\mathbb{S}}$ also includes delays for all time-graded modalities $\langle \tau \rangle$ in $\mathbb{S}$. We define $\mathbb{K}_{\mathbb{S}}$ as follows:

$$\mathbb{K}_{\mathbb{S}} \stackrel{\text{def}}{=} \begin{cases} [\,], & \text{if } \mathbb{S} = \emptyset \\ \mathbb{K}_{\mathbb{S}'}[\mathsf{delay}\ \tau\ [\,]], & \text{if } \mathbb{S} = \mathbb{S}', \langle \tau \rangle \\ \mathbb{K}_{\mathbb{S}'}[\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ [\,]], & \text{if } \mathbb{S} = \mathbb{S}', x \mapsto_{[\tau]X} V \end{cases}$$

These computation term contexts $\mathbb{K}_{\mathbb{S}}$ are related to the states $\mathbb{S}$ in natural and expected ways.

Proposition 4.5.

1. *For all* $\mathbb{S}$ *and* $\mathbb{S}'$*, we have* $\mathbb{K}_{\mathbb{S},\mathbb{S}'} = \mathbb{K}_{\mathbb{S}}[\mathbb{K}_{\mathbb{S}'}]$ *and* $\Gamma_{\mathbb{K}_{\mathbb{S}}} = \Gamma_{\mathbb{S}}$.
2. *If* $\Gamma \vdash \mathbb{S}$*, then we have* $\Gamma \vdash \mathbb{K}_{\mathbb{S}} : \tau_{\mathbb{S}}$.
3. *If* $\mathbb{S} = \mathbb{S}', x \mapsto_{[\tau]X} V, \mathbb{S}''$*, then we have* $\mathbb{K}_{\mathbb{S}} = \mathbb{K}_{\mathbb{S}'}[\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K}_{\mathbb{S}''}]$.

4.3.2 Evaluation Contexts

The second ingredient we need for stating and proving the equational soundness theorem is *evaluation contexts* $\mathbb{E}$. We use them below to have a stronger induction hypothesis. They are defined as follows:

$$\mathbb{E} ::= [\,] \mid \mathsf{let}\ x = \mathbb{E}\ \mathsf{in}\ N \mid \mathsf{handle}\ \mathbb{E}\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N$$

We also identify *well-formed evaluation contexts* $\Gamma \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$, given by rules analogous to the typing rules for the corresponding computations, so we omit them. Here, the subscript $[X \,!\, \tau]$ in the judgement $\Gamma \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$ denotes the type of computations that we can fill the hole $[\,]$ in $\mathbb{E}$ with, giving us:

Proposition 4.6. *If* $\Gamma \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$ *and* $\Gamma \vdash M : X \,!\, \tau$*, then* $\Gamma \vdash \mathbb{E}[M] : Y \,!\, \tau'$.

Proposition 4.7. *If* $\Gamma \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$ *and* $\Gamma \vdash M \equiv N : X \,!\, \tau$*, then* $\Gamma \vdash \mathbb{E}[M] \equiv \mathbb{E}[N] : Y \,!\, \tau'$.

As each $\mathbb{E}$ is a sequence of lets and handles, we can prove algebraicity equations for them.

Proposition 4.8. *If* $\Gamma \vdash_{[Y!\tau']} \mathbb{E} : Z \,!\, \tau''$ *and* $\Gamma - \tau \vdash V : [\tau]X$ *and* $\Gamma, x{:}X \vdash N : Y \,!\, \tau'$*, then we have the equation* $\Gamma \vdash \mathbb{E}[\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N] \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{E}[N] : Z \,!\, \tau''$*, and similarly for* box *and* delay.

4.3.3 Equational Soundness

Soundness then takes the following form, using evaluation contexts for a stronger induction hypothesis.

Theorem 4.9. *If* $\vdash \mathbb{S}$ *and* $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$*, and if* $\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle$*, for some* $\mathbb{S}'$ *and* M'*, with* $\mathbb{S}' = \mathbb{S}, \mathbb{S}''$*, then for every evaluation context* $\Gamma_{\mathbb{S}} \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$*, we have* $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[M]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[M']]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$.

Proof. We proceed by induction on the derivation of $\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle$. In all cases, we use Thm. 3.10 to prove that $\tau_{\mathbb{S}} + \tau = \tau_{\mathbb{S}'} + \tau''$ and $\Gamma_{\mathbb{S}'} \vdash M' : X \,!\, \tau''$, for some τ''. In addition, we use Prop. 3.8 to recover and use the well-typedness information for the subcomputations of M and M'.

Cases for reduction rules that do not change state and for which there are corresponding equations (Sem-App, Sem-Match, Sem-Let-Ret, Sem-Let-Op, Sem-Handle-Ret, Sem-Handle-Op) are proved as follows: In these cases, $\mathbb{S}' = \mathbb{S}$ and $\mathbb{S}'' = \emptyset$, and thus $\mathbb{K}_{\mathbb{S}''} = \mathbb{K}_{\emptyset} = [\,]$. By choosing the equation corresponding to the given reduction rule, we get $\Gamma_{\mathbb{S}} \vdash M \equiv N : X \,!\, \tau$. Next, given any $\Gamma_{\mathbb{S}} \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$, we apply Prop. 4.7 to get the equation $\Gamma_{\mathbb{S}} \vdash \mathbb{E}[M] \equiv \mathbb{E}[N] : Y \,!\, \tau'$. Finally, applying Prop. 4.3, we get the required equation $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[M]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[N]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[N]]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$.

Sem-Let-Cong: Here, $M = \mathsf{let}\ x = N\ \mathsf{in}\ P$ and $M' = \mathsf{let}\ x = N'\ \mathsf{in}\ P$. Applying Prop. 3.8 to $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$, we get $\tau = \tau_1 + \tau_2$, and that N and P are typed as $\Gamma_{\mathbb{S}} \vdash N : X' \,!\, \tau_1$ and $\Gamma_{\mathbb{S}}, \langle \tau_1 \rangle, x : X' \vdash P : X \,!\, \tau_2$, for some X'. Using the induction hypothesis on $\Gamma_{\mathbb{S}} \vdash N : X' \,!\, \tau_1$ and $\langle \mathbb{S} \mid N \rangle \rightsquigarrow \langle \mathbb{S}' \mid N' \rangle$, we get for every $\Gamma_{\mathbb{S}} \vdash_{[X'!\tau_1]} \mathbb{E}' : Z \,!\, \tau''$ the equation $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}'[N]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}'[\mathbb{K}_{\mathbb{S}''}[N']]] : Z \,!\, (\tau_{\mathbb{S}} + \tau'')$. To prove the required equation, for every evaluation context $\Gamma_{\mathbb{S}} \vdash_{[X!\tau]} \mathbb{E} : Y \,!\, \tau'$, we apply the above result with the evaluation context $\mathbb{E}' \stackrel{\text{def}}{=} \mathbb{E}[\mathsf{let}\ x = [\,]\ \mathsf{in}\ P]$, giving us the equation $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathsf{let}\ x = N\ \mathsf{in}\ P]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathsf{let}\ x = \mathbb{K}_{\mathbb{S}''}[N']\ \mathsf{in}\ P]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$. Finally, we apply the algebraicity of computation term contexts from Prop. 4.4, getting $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[M]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathsf{let}\ x = N\ \mathsf{in}\ P]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[\mathsf{let}\ x = N'\ \mathsf{in}\ P]]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[M']]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$.

We make two useful observations about this case. First, the computation in the right-hand side of the equation is well-typed because Prop. 3.1 and Thm. 3.10 ensure that $\tau_{\mathbb{S}} + \tau_1 = \tau_{\mathbb{S}'} + \tau_1' = \tau_{\mathbb{S}} + \tau_{\mathbb{S}''} + \tau_1'$, meaning that $\tau_1 = \tau_{\mathbb{S}''} + \tau_1'$, with $\Gamma_{\mathbb{S}}, \Gamma_{\mathbb{S}''} \vdash N' : X' \,!\, \tau_1'$, and therefore $\Gamma_{\mathbb{S}} \vdash \mathbb{K}_{\mathbb{S}''}[N'] : X' \,!\, \tau_1$. Second, the use of evaluation contexts is crucial to be able to lift the equation relating N and N' that we get from the induction hypothesis to an equation relating the computations $\mathsf{let}\ x = N\ \mathsf{in}\ P$ and $\mathsf{let}\ x = N'\ \mathsf{in}\ P$.

Sem-Handle-Cong: Analogous to Sem-Let-Cong, but with $\mathbb{E}' \stackrel{\text{def}}{=} \mathbb{E}[\mathsf{handle}\ [\,]\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ P]$.

Sem-Delay: Here, $M = \mathsf{delay}\ \tau_1\ M'$ and $\mathbb{S}' = \mathbb{S}, \langle \tau_1 \rangle$, and thus $\mathbb{S}'' = \emptyset, \langle \tau_1 \rangle$. Applying Prop. 3.8 to $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$, we get $\tau = \tau_1 + \tau_2$ and $\Gamma_{\mathbb{S}}, \langle \tau_1 \rangle \vdash M' : X \,!\, \tau_2$. We observe that $\mathbb{K}_{\mathbb{S}''} = \mathbb{K}_{(\emptyset, \langle \tau_1 \rangle)} = \mathsf{delay}\ \tau_1\ [\,]$. Thus, we get the required equation as $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[M]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathsf{delay}\ \tau_1\ M']] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[M']]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$.

Sem-Box: Here, $M = \mathsf{box}_{[\tau'']X'}\ V\ \mathsf{as}\ x\ \mathsf{in}\ M'$ and $\mathbb{S}' = \mathbb{S}, x \mapsto_{[\tau'']X'} V$, thus $\mathbb{S}'' = \emptyset, x \mapsto_{[\tau'']X'} V$. Applying Prop. 3.8, we get $\Gamma_{\mathbb{S}}, x : [\tau'']X' \vdash M' : [X]\,\tau$. We observe that $\mathbb{K}_{\mathbb{S}''} = \mathbb{K}_{(\emptyset, x \mapsto_{[\tau'']X'} V)} = \mathsf{box}_{[\tau'']X'}\ V\ \mathsf{as}\ x\ \mathsf{in}\ [\,]$. Thus, we get the equation $\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[M]] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathsf{box}_{[\tau'']X'}\ V\ \mathsf{as}\ x\ \mathsf{in}\ M']] \equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[M']]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$.

Sem-Unbox: Here, $M = \mathsf{unbox}_{[\tau'']X'}\ y\ \mathsf{as}\ x\ \mathsf{in}\ N$ and $M' = N[\mathbb{S}[y]/x]$ and $\mathbb{S}' = \mathbb{S}$, and thus $\mathbb{S}'' = \emptyset$. Applying Prop. 3.8 to $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$, we get that $y : [\tau'']X' \in \Gamma_{\mathbb{S}} - \tau''$ and $\Gamma_{\mathbb{S}}, x : X' \vdash N : [X]\,\tau$. By reasoning analogously to the proof of Thm. 3.10, we get for $\Gamma_{\mathbb{S}} = (\Gamma_{\mathbb{S}})_{y,1}, y : [\tau'']X', (\Gamma_{\mathbb{S}})_{y,2}$ that $\tau'' \leq \tau_{(\Gamma_{\mathbb{S}})_{y,2}}$ and $\Gamma_{\mathbb{S}} \vdash \mathbb{S}[y] : X'$. We further observe that $\mathbb{S} = \mathbb{S}_{y,1}, y \mapsto_{[\tau'']X'} \mathbb{S}[y], \mathbb{S}_{y,2}$ and $(\Gamma_{\mathbb{S}})_{y,i} = \Gamma_{\mathbb{S}_{y,i}}$. Putting these observations together, we can prove the required equation as the following sequence of equations:

$$
\begin{aligned}
&\vdash \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathsf{unbox}_{[\tau'']X'}\ y\ \mathsf{as}\ x\ \mathsf{in}\ N]] \\
&\equiv \mathbb{K}_{\mathbb{S}}[\mathsf{unbox}_{[\tau'']X'}\ y\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{E}[N]] && \text{(Props. 4.3 and 4.8)} \\
&\equiv \mathbb{K}_{\mathbb{S}_{y,1}}[\mathsf{box}_{[\tau'']X'}\ \mathbb{S}[y]\ \mathsf{as}\ y\ \mathsf{in}\ \mathbb{K}_{\mathbb{S}_{y,2}}[\mathsf{unbox}_{[\tau'']X'}\ y\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{E}[N]]] && \text{(Prop. 4.5)} \\
&\equiv \mathbb{K}_{\mathbb{S}_{y,1}}[\mathsf{box}_{[\tau'']X'}\ \mathbb{S}[y]\ \mathsf{as}\ y\ \mathsf{in}\ \mathbb{K}_{\mathbb{S}_{y,2}}[\mathbb{E}[N[\mathbb{S}[y]/x]]]] && \text{(interaction eq. and } \tau'' \leq \tau_{(\Gamma_{\mathbb{S}})_{y,2}} = \tau_{\mathbb{K}_{\mathbb{S}_{y,2}}})
\end{aligned}
$$

$\equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[N[\mathbb{S}[y]/x]]]$ (Props. 4.3 and 4.8)

$\equiv \mathbb{K}_{\mathbb{S}}[\mathbb{E}[\mathbb{K}_{\mathbb{S}''}[M']]] : Y \,!\, (\tau_{\mathbb{S}} + \tau')$ ($M' = N[\mathbb{S}[y]/x]$ and $\mathbb{K}_{\mathbb{S}''} = \mathbb{K}_{\emptyset} = [\,]$) □

When choosing $\mathbb{E} \stackrel{\text{def}}{=} [\,]$, and using Prop. 4.5 for $\mathbb{K}_{\mathbb{S}}[\mathbb{K}_{\mathbb{S}'}] = \mathbb{K}_{\mathbb{S}'}$, we get the desired soundness result.

Corollary 4.10. *If* $\vdash \mathbb{S}$ *and* $\Gamma_{\mathbb{S}} \vdash M : X \,!\, \tau$ *and* $\langle \mathbb{S} \mid M \rangle \rightsquigarrow \langle \mathbb{S}' \mid M' \rangle$*, then* $\vdash \mathbb{K}_{\mathbb{S}}[M] \equiv \mathbb{K}_{\mathbb{S}'}[M'] : X \,!\, (\tau_{\mathbb{S}} + \tau)$.

5 Related Work

Our use of a stateful operational semantics for describing the computational behaviour of $\lambda_{[\tau]}$ is reminiscent of Krishnaswami's semantics for a modally typed functional reactive programming (FRP) language [13]. In Krishnaswami's work, states contain pointers to values available either now or later, after some number of clock ticks. These stored (pointers to) values are analogous to the resources $x \mapsto_{[\tau]X} V$ that we store in our states. However, whereas we model time passage by extending the context with context modalities, Krishnaswami has a separate (clock) tick relation that moves states forward in time. The consequence of this is that values available now are not necessarily available in future states in Krishnaswami's semantics, while our semantics allows us to unbox values that were boxed long ago.

As noted earlier, the equational theory of Sect. 4.2 is closely related to and inspired by the standard equational presentations of local state [19, 21, 26]. It differs from those presentations by not including an assignment operation and by having the box-unbox interaction equation apply at a distance, and generally not when box and unbox are sequenced one after another. Another difference with these theories of local state is that the type X in our modal type $[\tau]X$ can be an arbitrary value type, whereas theories of local state typically restrict reference types to base types, meaning that in $\lambda_{[\tau]}$ we are in fact dealing with higher-order (local) state [3, 15]. If we were to restrict X to ground types, we would have an instance of full ground state [9], as resource types could nest modal types but not combine them with functions.

6 Conclusion

In this paper we have proposed a stateful time-aware operational semantics for $\lambda_{[\tau]}$, a core calculus for programming with temporal resources. The semantics uses states to track which temporal resources are used by the program and to also provide a cost model by tracking the run time of computations. This gives us an alternative, dynamic means for analysing the use of temporal resources, complementing $\lambda_{[\tau]}$'s time-graded, modal type-and-effect system. We proved this semantics to be type-safe, and we also proved it sound with respect to a temporal variant of standard equational presentations of local state.

Our soundness result is admittedly a bit of a stopgap measure—while it shows that our semantics agrees with a well-understood notion of computation, eventually it would be desirable to prove a more standard soundness result against a denotational semantics modelling the equational theory we consider. For an even stronger guarantee, it would be desirable to also prove our semantics adequate with respect to such a denotational semantics. For adequacy, the states appearing in our operational semantics likely need to be quotiented a little, e.g., so as to not distinguish between the order of resources stored between two occurrences of the context modality, and to not distinguish between $\mathbb{S}, \langle \tau \rangle, \langle \tau' \rangle$ and $\mathbb{S}, \langle \tau + \tau' \rangle$.

Acknowledgements This paper is based on the second author's MSc thesis (in Slovene) [30] defended at the Faculty of Mathematics and Physics at the University of Ljubljana (UL FMF). The first author was employed at UL FMF while majority of this work was conducted. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-21-1-0024. We are grateful to Matija Pretnar for many useful discussions and comments about this work.

References

[1] Danel Ahman (2023): *When Programs Have to Watch Paint Dry*. In Orna Kupferman & Pawel Sobocinski, editors: *Foundations of Software Science and Computation Structures - 26th International Conference, FoSSaCS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings*, *Lecture Notes in Computer Science* 13992, Springer, pp. 1–23, doi:10.1007/978-3-031-30829-1_1.

[2] Danel Ahman & Andrej Bauer (2020): *Runners in Action*. In Peter Müller, editor: *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, *Lecture Notes in Computer Science* 12075, Springer, pp. 29–55, doi:10.1007/978-3-030-44914-8_2.

[3] Amal Ahmed (2004): *Semantics of types for mutable state*. Ph.D. thesis, Princeton University, USA.

[4] P. N. Benton, Gavin M. Bierman, Valeria de Paiva & Martin Hyland (1992): *Linear Lambda-Calculus and Categorial Models Revisited*. In Egon Börger, Gerhard Jäger, Hans Kleine Büning, Simone Martini & Michael M. Richter, editors: *Computer Science Logic, 6th Workshop, CSL '92, San Miniato, Italy, September 28 - October 2, 1992, Selected Papers*, *Lecture Notes in Computer Science* 702, Springer, pp. 61–84, doi:10.1007/3-540-56992-8_6.

[5] Ranald Clouston (2018): *Fitch-Style Modal Lambda Calculi*. In Christel Baier & Ugo Dal Lago, editors: *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, *Lecture Notes in Computer Science* 10803, Springer, pp. 258–275, doi:10.1007/978-3-319-89366-2_14.

[6] Jean-Yves Girard (1987): *Linear logic*. *Theoretical Computer Science* 50(1), pp. 1–101, doi:10.1016/0304-3975(87)90045-4.

[7] Daniel Gratzer, Evan Cavallo, G. A. Kavvos, Adrien Guatto & Lars Birkedal (2022): *Modalities and Parametric Adjoints*. *ACM Trans. Comput. Log.* 23(3), pp. 18:1–18:29, doi:10.1145/3514241.

[8] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In Chris Hankin, editor: *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, *Lecture Notes in Computer Science* 1381, Springer, pp. 122–138, doi:10.1007/BFB0053567.

[9] Ohad Kammar, Paul Blain Levy, Sean K. Moss & Sam Staton (2017): *A monad for full ground reference cells*. In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, IEEE Computer Society, pp. 1–12, doi:10.1109/LICS.2017.8005109.

[10] Ohad Kammar, Sam Lindley & Nicolas Oury (2013): *Handlers in action*. In Greg Morrisett & Tarmo Uustalu, editors: *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, ACM, pp. 145–158, doi:10.1145/2500365.2500590.

[11] Shin-ya Katsumata (2014): *Parametric effect monads and semantics of effect systems*. In Suresh Jagannathan & Peter Sewell, editors: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, ACM, pp. 633–646, doi:10.1145/2535838.2535846.

[12] P. Koopman, J. Fokker, S. Smetsers, M. van Eekelen & R. Plasmeijer (1999): *Functional Programming in Clean*. University of Nijmegen. Draft.

[13] Neelakantan R. Krishnaswami (2013): *Higher-order functional reactive programming without spacetime leaks*. In Greg Morrisett & Tarmo Uustalu, editors: *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, ACM, pp. 221–232, doi:10.1145/2500365.2500588.

[14] Daan Leijen (2018): *Algebraic Effect Handlers with Resources and Deep Finalization*. Technical Report MSR-TR-2018-10. Available at `https://www.microsoft.com/en-us/research/publication/algebraic-effect-handlers-resources-deep-finalization/`.

[15] Paul Blain Levy (2002): *Possible World Semantics for General Storage in Call-By-Value*. In Julian C. Bradfield, editor: *Computer Science Logic, 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL, Edinburgh, Scotland, UK, September 22-25, 2002, Proceedings*, *Lecture Notes in Computer Science* 2471, Springer, pp. 232–246, doi:10.1007/3-540-45793-3_16.

[16] Paul Blain Levy (2004): *Call-By-Push-Value: A Functional/Imperative Synthesis*. *Semantics Structures in Computation* 2, Springer, doi:10.1007/978-94-007-0954-6.

[17] Paul Blain Levy, John Power & Hayo Thielecke (2003): *Modelling environments in call-by-value programming languages*. *Inf. Comput.* 185(2), pp. 182–210, doi:10.1016/S0890-5401(03)00088-9.

[18] P.-A. Melliès (2012): *Parametric Monads and Enriched Adjunctions*. Manuscript. `https://www.irif.fr/~mellies/tensorial-logic/8-parametric-monads-and-enriched-adjunctions.pdf`.

[19] Paul-André Melliès (2014): *Local States in String Diagrams*. In Gilles Dowek, editor: *Rewriting and Typed Lambda Calculi*, Springer International Publishing, Cham, pp. 334–348.

[20] Eugenio Moggi (1991): *Notions of Computation and Monads*. *Inf. Comput.* 93(1), pp. 55–92, doi:10.1016/0890-5401(91)90052-4.

[21] Gordon D. Plotkin & John Power (2002): *Notions of Computation Determine Monads*. In Mogens Nielsen & Uffe Engberg, editors: *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, *Lecture Notes in Computer Science* 2303, Springer, pp. 342–356, doi:10.1007/3-540-45931-6_24.

[22] Gordon D. Plotkin & John Power (2003): *Algebraic Operations and Generic Effects*. *Appl. Categorical Struct.* 11(1), pp. 69–94, doi:10.1023/A:1023064908962.

[23] Gordon D. Plotkin & Matija Pretnar (2013): *Handling Algebraic Effects*. *Log. Methods Comput. Sci.* 9(4), doi:10.2168/LMCS-9(4:23)2013.

[24] John C. Reynolds (2002): *Separation Logic: A Logic for Shared Mutable Data Structures*. In: *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, IEEE Computer Society, pp. 55–74, doi:10.1109/LICS.2002.1029817.

[25] A. L. Smirnov (2008): *Graded monads and rings of polynomials*. *Journal of Mathematical Sciences* 151(3), pp. 3032–3051, doi:10.1007/s10958-008-9013-7.

[26] Sam Staton (2010): *Completeness for Algebraic Theories of Local State*. In C.-H. Luke Ong, editor: *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, *Lecture Notes in Computer Science* 6014, Springer, pp. 48–63, doi:10.1007/978-3-642-12032-9_5.

[27] The Agda Team (2024): *The Agda Wiki*. Available at `https://wiki.portal.chalmers.se/agda/pmwiki.php`.

[28] Philip Wadler (2014): *Propositions as sessions*. *J. Funct. Program.* 24(2-3), pp. 384–418, doi:10.1017/S095679681400001X.

[29] Andrew K. Wright & Matthias Felleisen (1994): *A Syntactic Approach to Type Soundness*. *Inf. Comput.* 115(1), pp. 38–94, doi:10.1006/INCO.1994.1093.

[30] Gašper Žajdela (2023): *Operacijska semantika za jezik s časovno omejenimi viri : magistrsko delo*. Available at `https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=eng&id=150138`. Faculty of Mathematics and Physics, University of Ljubljana.

A Equational Theory

Values and computations both have an equality judgement, $\Gamma \vdash V \equiv W : X$ and $\Gamma \vdash M \equiv N : X \,!\, \tau$, respectively. We presuppose that we only compare well-typed values and computations. For most part the contexts and types play no significant role in equations, so we omit them wherever convenient.

The rules defining these judgements are given in Figs. 6 to 9. We have split these rules between different fragments of $\lambda_{[\tau]}$. We omit standard equations which specify how substitution is performed, and equations stating that these equality relations are congruences with respect to all the term formers.

Unit Type

$$V \equiv () \qquad (\text{where } \Gamma \vdash V : \mathsf{unit})$$

Product Type

$$\begin{aligned} \mathsf{match}\ (V,W)\ \mathsf{with}\ \{(x,y) \mapsto N\} &\equiv N[V/x, W/y] \\ M[V/Z] &\equiv \mathsf{match}\ V\ \mathsf{with}\ \{(x,y) \mapsto M[(x,y)/z]\} \qquad (\text{where } \Gamma \vdash V : X \times Y) \end{aligned}$$

Function Type

$$\begin{aligned} (\mathsf{fun}\ (x{:}X) \mapsto M)\, V &\equiv M[V/x] \\ V &\equiv \mathsf{fun}\ (x{:}X) \mapsto V\, x \qquad (\text{where } \Gamma \vdash V : X \to Y \,!\, \tau) \end{aligned}$$

Figure 6: Non-modal fragment of the equational theory of $\lambda_{[\tau]}$.

Return Values

$$\mathsf{let}\ x = (\mathsf{return}\ V)\ \mathsf{in}\ N \equiv N[V/x] \qquad \mathsf{handle}\ (\mathsf{return}\ V)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv N[V/z]$$

Algebraicity

$$\mathsf{let}\ x = (\mathsf{op}\ V\ (y\ .\ M))\ \mathsf{in}\ N \equiv \mathsf{op}\ V\ (y\ .\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)) \qquad (y \notin fv(N))$$

Effect Handling

$$\begin{aligned} &\mathsf{handle}\ (\mathsf{op}\ V\ (y\ .\ M))\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv && (H = (x\ .\ k\ .\ M_{\mathsf{op}})_{\mathsf{op} \in \mathscr{O}}) \\ &\quad \mathsf{box}\ \big(\mathsf{fun}\ (y : B_{\mathsf{op}}) \mapsto \mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N\big)\ \mathsf{as}\ w\ \mathsf{in}\ M_{\mathsf{op}}[V/x, w/k] && (y \notin fv(H), y \notin fv(N)) \end{aligned}$$

Associativity

$$\mathsf{let}\ x = (\mathsf{let}\ y = M\ \mathsf{in}\ N)\ \mathsf{in}\ P \equiv \mathsf{let}\ y = M\ \mathsf{in}\ (\mathsf{let}\ x = N\ \mathsf{in}\ P) \qquad (y \notin fv(P))$$

Figure 7: return, let, and handle fragment of the equational theory of $\lambda_{[\tau]}$.

Algebraicity

$$\mathsf{let}\ x = (\mathsf{delay}\ \tau\ M)\ \mathsf{in}\ N \equiv \mathsf{delay}\ \tau\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)$$
$$\mathsf{handle}\ (\mathsf{delay}\ \tau\ M)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv \mathsf{delay}\ \tau\ (\mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N)$$

Time Grades

$$\mathsf{delay}\ 0\ M \equiv M$$
$$\mathsf{delay}\ \tau\ (\mathsf{delay}\ \tau'\ M) \equiv \mathsf{delay}\ (\tau + \tau')\ M$$

Figure 8: delay fragment of the equational theory of $\lambda_{[\tau]}$.

Algebraicity

$$\mathsf{let}\ x = (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{in}\ N \equiv \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)$$
$$\mathsf{let}\ x = (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{in}\ N \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{let}\ x = M\ \mathsf{in}\ N)$$
$$\mathsf{handle}\ (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N)$$
$$\mathsf{handle}\ (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ M)\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{handle}\ M\ \mathsf{with}\ H\ \mathsf{to}\ z\ \mathsf{in}\ N)$$
(with $y \notin fv(N)$ in all four equations)

Commutativity

$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{box}_{[\tau']Y}\ W\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{box}_{[\tau']Y}\ W\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N)$$
$$\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N)$$
$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{unbox}_{[\tau']X'}\ W\ \mathsf{as}\ y\ \mathsf{in}\ (\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N)$$
(with $x \notin fv(W), y \notin fv(V)$ in all three equations)

Interaction

$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K}[\mathsf{unbox}_{[\tau]X}\ x\ \mathsf{as}\ y\ \mathsf{in}\ N] \equiv \mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ \mathbb{K}[N[V/y]] \qquad (\tau_{\mathbb{K}} \geq \tau)$$

Discarding and Contraction

$$\mathsf{box}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N \equiv N \qquad (x \notin fv(N))$$
$$\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N \equiv N \qquad (x \notin fv(N))$$
$$\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ (\mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ y\ \mathsf{in}\ N) \equiv \mathsf{unbox}_{[\tau]X}\ V\ \mathsf{as}\ x\ \mathsf{in}\ N[x/y]$$

Figure 9: box and unbox fragment of the equational theory of $\lambda_{[\tau]}$.