Destination calculus

A linear λ–calculus for pure, functional memory updates

ARNAUD SPIWACK, Modus Create, France
THOMAS BAGREL, LORIA/Inria, France and Modus Create, France

We present the destination calculus, a linear λ–calculus for pure, functional memory updates. We introduce the syntax, type system, and operational semantics of the destination calculus, and prove type safety formally in the Coq proof assistant.

We show how the principles of the destination calculus can form a theoretical ground for destination-passing style programming in functional languages. In particular, we detail how the present work can be applied to Linear Haskell to lift the main restriction of DPS programming in Haskell as developed in [1]. We illustrate this with a range of pseudo-Haskell examples.

ACM Reference Format:
Arnaud Spiwack and Thomas Bagrel. 2024. Destination calculus: A linear λ–calculus for pure, functional memory updates. In . ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Contents

POPL'25, January 19 – 25, 2025, Denver, Colorado

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

2 SYSTEM IN ACTION ON SIMPLE EXAMPLES

Build up to DList.

3 LIMITIONS OF THE PREVIOUS APPROACH

3.1 Breadth-first tree traversal

3.2 Storing linear data in destination-based data structures

3.3 Need for scope control

4 UPDATED BREADTH-FIRST TREE TRAVERSAL

5 LANGUAGE SYNTAX

5.1 Introducing the *ampar*

Minamide's work[3] is the earliest record we could find of a functional calculus integrating the idea of incomplete data structures (structures with holes) that exist as first class values and can be interacted with by the user.

In that paper, a structure with a hole is named *hole abstraction*. In the body of a hole abstraction, the bound *hole variable* should be used linearly (exactly once), and must only be used as a parameter of a data constructor. In other terms, the bound *hole variable* cannot be pattern-matched on or used as a parameter of a function call. A hole abstraction is thus a weak form of linear lambda abstraction, which just moves a piece of data into a bigger data structure.

In fact, the type of hole abstraction $(\mathsf{T}_1, \mathsf{T}_2)\mathsf{hfun}$ in Minamine's work shares a lot of similarity with the separating implication or *magic wand* $\mathsf{T}_1 \mathrel{-\!\!*} \mathsf{T}_2$ from separation logic: given a piece of memory matching description T_1, we obtain a (complete) piece of memory matching description T_2.

Now, in classical linear logic, we know we can transform linear implication $\mathsf{T}_1 \multimap \mathsf{T}_2$ into $\mathsf{T}_1{}^{\perp} \parr \mathsf{T}_2$. Doing so for the *wand* type $(\mathsf{T}_1, \mathsf{T}_2)\mathsf{hfun}$ or $\mathsf{T}_1 \mathrel{-\!\!*} \mathsf{T}_2$ gives $\lfloor \mathsf{T}_1 \rfloor \mathbin{\widehat{\parr}} \mathsf{T}_2$, where $\lfloor \cdot \rfloor$ is memory negation, and $\widehat{\parr}$ is a memory *par* (weaker than the CLL *par* that allows more "interaction" of its two sides).

Transforming the hole abstraction from its original implication form to a *par* form let us consider the type $\lfloor \mathsf{T}_1 \rfloor$ of *sink* or *destination* of T_1 as a first class component of our calculus. We also get to see the hole abstraction aka memory par as a pair-like structure, where the two sides might be coupled together in a way that prevent using both of them simultaneously.

From memory par $\widehat{\parr}$ *to ampar* $\ltimes$. In CLL, the cut rule states that given $\mathsf{T}_1 \parr \mathsf{T}_2$, we can free up T_1 by providing an eliminator of T_2, or free up T_2 by providing an eliminator of T_1. The eliminator of T can be $\mathsf{T}^{\perp}$, or $\mathsf{T}^{\perp^{-1}} = \mathsf{T}'$ if T is already of the form $\mathsf{T}'^{\perp}$. In a classical setting, thanks to the involutive nature of negation $\cdot^{\perp}$, the two potential forms of the eliminator of T are equal.

In destination calculus though, we don't have an involutive memory negation $\lfloor \cdot \rfloor$. If we are provided with a destination of destination $\to\!\mathsf{h}' : \lfloor \lfloor \mathsf{T} \rfloor \rfloor$, we know that some structure is expecting to store a destination of type $\lfloor \mathsf{T} \rfloor$. If ever that structure is consumed, then the destination stored inside will have to be filled with a value (remember we are in a linear calculus). So if we allocate a new memory slot of type $\mathsf{h} : \mathsf{T}$ and its linked destination $\to\!\mathsf{h} : \lfloor \mathsf{T} \rfloor$, and write $\to\!\mathsf{h}$ to the memory slot pointed to by $\to\!\mathsf{h}'$, then we can get back a value of type T at h if ever the structure pointed to by $\to\!\mathsf{h}'$ is consumed. Thus, a destination of destination is only equivalent to the promise of an eventual value, not an immediate usable one.

As a result, in destination calculus, we cannot have the same kind of cut rule as in CLL. This is, in fact, the part of destination calculus that was the hardest to design, and the source of a lot of early errors. For a destination of type $\lfloor \mathsf{T} \rfloor$, both storing it through a destination of destination $\lfloor \lfloor \mathsf{T} \rfloor \rfloor$ or using it to store a value of type T constitute a linear use of the destination. But only the latter is a genuine consumption in the sense that it guarantees that the hole associated to the destination has been filled! Storing away the destination of type $\lfloor \mathsf{T} \rfloor$ in $\mathsf{T} \mathbin{\widehat{\parr}} \lfloor \mathsf{T} \rfloor$ (through a destination of destination of type $\lfloor \lfloor \mathsf{T} \rfloor \rfloor$) should not allow to free up the T, as it would in a CLL-like setting.

5.2 Names and variables

The destination calculus uses two classes of names: regular (meta) variable names x, y, and hole names, h, h_1, h_2 which represents the identifier or address of a memory cell that hasn't been written to yet.

$\mathsf{var}, \mathsf{x}, \mathsf{y}, \mathsf{d}, \mathsf{un}, \mathsf{ex}, \mathsf{st}$ Variable names

$$
\begin{array}{llll}
\mathsf{hvar},\ \mathsf{h} & ::= & & \text{Hole (or destination) name, represented by a natural number} \\
& \mid & \mathsf{h}{+}\mathsf{h}' & \mathsf{M} \\
& \mid & \mathsf{h}[\mathsf{H}{\pm}\mathsf{h}'] & \mathsf{M} \quad \text{Shift by } \mathsf{h}' \text{ if } \mathsf{h} \in \mathsf{H} \\
& \mid & \mathit{max}(\mathsf{H}) & \mathsf{M} \quad \text{Maximum of a set of hole names}
\end{array}
$$

Hole names are represented by natural numbers under the hood, so they can act both as relative offsets or absolute positions in memory. Typically, when a structure is effectively allocated, its hole names are shifted by the maximum hole name encountered so far in the program ; this corresponds to finding the next unused memory cell in which to write new data.

We sometimes need to keep track of hole names bound by a particular runtime value or evaluation context, hence we also define sets of hole names $\mathsf{H}, \mathsf{H}_1, \mathsf{H}_2 \ldots$.

$$
\begin{array}{llll}
\mathsf{hvars},\ \mathsf{H} & ::= & & \text{Set of hole names} \\
& \mid & \{\mathsf{h}_1, .., \mathsf{h}_k\} & \\
& \mid & \mathsf{H}_1 \cup \mathsf{H}_2 & \mathsf{M} \quad \text{Union of sets} \\
& \mid & \mathsf{H}{\pm}\mathsf{h}' & \mathsf{M} \quad \text{Shift all names from } \mathsf{H} \text{ by } \mathsf{h}'. \\
& \mid & \mathit{hvars}(\Gamma) & \mathsf{M} \quad \text{Hole names bound by the typing context } \Gamma \\
& \mid & \mathit{hvars}(C) & \mathsf{M} \quad \text{Hole names bound by the evaluation context } C
\end{array}
$$

Shifting all hole names in a set by a given offset h' is denoted $\mathsf{H}{\pm}\mathsf{h}'$. We also define a conditional shift operation $[\mathsf{H}{\pm}\mathsf{h}']$ which shifts each hole name appearing in the operand to the left of the brackets by h' if this hole name is also member of H. This conditional shift can be used on a single hole name, a value, or a typing context.

5.3 Term and value core syntax

Destination calculus is based on linear simply-typed λ-calculus, with built-in support for sums, pairs, and exponentials. The syntax of terms is quite unusual, as we need to introduce all the tooling required to manipulate destinations, which constitute the primitive way of building a data structures for the user.

In fact, the grammatical class of values v, presented as a subset of terms t, could almost be removed completely from the user syntax, and just used as a denotation for runtime data structures. We only need to keep the *ampar* value $_{\{\mathsf{h}\}}\langle \mathsf{h}\ {}_\wedge\!\!\rightarrow\!\mathsf{h} \rangle$ as part of the user syntax as a way to spawn a fresh memory cell to be later filled using destination-filling primitives (see `alloc` in Section 5.4).

$$
\begin{array}{llll}
\mathit{term},\ t,\ u & ::= & & \text{Term} \\
& \mid & v & \text{Value} \\
& \mid & \mathsf{x} & \text{Variable} \\
& \mid & t \rhd t' & \text{Application} \\
& \mid & t \mathbin{;} u & \text{Pattern-match on unit} \\
& \mid & t \rhd \mathtt{case}_{\mathsf{m}}\ \{\mathtt{Inl}\ \mathsf{x}_1 \mapsto u_1\ ,\ \mathtt{Inr}\ \mathsf{x}_2 \mapsto u_2\} & \text{Pattern-match on sum} \\
& \mid & t \rhd \mathtt{case}_{\mathsf{m}}\ (\mathsf{x}_1\ ,\ \mathsf{x}_2) \mapsto u & \text{Pattern-match on product} \\
& \mid & t \rhd \mathtt{case}_{\mathsf{m}}\ \mathtt{E}_{\mathsf{n}}\ \mathsf{x} \mapsto u & \text{Pattern-match on exponential} \\
& \mid & t \rhd \mathtt{map}\ \mathsf{x} \mapsto t' & \text{Map over the right side of ampar} \\
& \mid & \mathtt{to}_{\ltimes}\ u & \text{Wrap into a trivial ampar} \\
& \mid & \mathtt{from}_{\ltimes}\ t & \text{Convert ampar to a pair} \\
& \mid & t \triangleleft () & \text{Fill destination with unit} \\
& \mid & t \triangleleft \mathtt{Inl} & \text{Fill destination with left variant}
\end{array}
$$

	$\mid$	$t \triangleleft \mathtt{Inr}$	Fill destination with right variant
	$\mid$	$t \triangleleft \mathtt{E}_{\mathrm{m}}$	Fill destination with exponential constructor
	$\mid$	$t \triangleleft (,)$	Fill destination with product constructor
	$\mid$	$t \triangleleft (\lambda x\,_{\mathrm{m}}\mapsto u)$	Fill destination with function
	$\mid$	$t \triangleleft\!\bullet\, t'$	Fill destination with root of other ampar
	$\mid$	$t[x \coloneqq v]$ M	

val, v	::=		Value
	$\mid$	h	Hole
	$\mid$	$\rightarrow h$	Destination
	$\mid$	$()$	Unit
	$\mid$	${}^{v}\lambda x\,_{\mathrm{m}}\mapsto u$	Function with no free variable
	$\mid$	$\mathtt{Inl}\ v$	Left variant for sum
	$\mid$	$\mathtt{Inr}\ v$	Right variant for sum
	$\mid$	$\mathtt{E}_{\mathrm{m}}\ v$	Exponential
	$\mid$	$(v_1\,,\,v_2)$	Product
	$\mid$	${}_{H}\langle v_2 \,{}_{\wedge}\, v_1 \rangle$	Ampar
	$\mid$	$v[H \pm h']$ M	Shift hole names inside v by h' if they belong to H.

Pattern-matching on every type of structure (except unit) is parametrized by a mode m to which the scrutinee is consumed. The variables which bind the subcomponents of the scrutinee then inherit this mode. In particular, this choice crystalize the equivalence $!_{\omega a}(T_1 \otimes T_2) \simeq (!_{\omega a}T_1)\otimes(!_{\omega a}T_2)$, which is not part of intuitionistic linear logic, but valid in Linear Haskell[2].

`map` is the main primitive to operate on an *ampar*, which represents an incomplete data structure whose building is in progress. `map` binds the right-hand side of the *ampar* — the one containing destinations of that *ampar* — to a variable, allowing those destinations to be operated on by destination-filling primitives. The left-hand side of the *ampar* is inaccessible as it is being mutated behind the scenes by the destination-filling primitives.

$\mathtt{to}_{\ltimes}$ embeds an already completed structure in an *ampar* whose left side is the structure, and right side is unit. We have an operator FillComp ($\triangleleft\!\bullet$) allowing to compose two *ampar*s by writing the root of the second one to a destination of the first one, so by throwing $\mathtt{to}_{\ltimes}$ to the mix, we can compose an *ampar* with a normal (completed) structure (see the sugar operator FillLeaf ($\triangleleft$) in Section 5.4).

$\mathtt{from}_{\ltimes}$ is used to convert an *ampar* to a pair, when the right side of the *ampar* is an exponential of the form $\mathtt{E}_{1\infty}\ v$. Indeed, when the right side has such form, it cannot contains destinations (as destinations always have a finite age), thus it cannot contain holes in its left side either (as holes on the left side are always compensated 1:1 by a destination on the right side). As a result, it is valid to convert an *ampar* to a pair in these circumstances. $\mathtt{from}_{\ltimes}$ is in particular used to extract a structure from its *ampar* building shell when it is complete (see the sugar operator $\mathtt{from}'_{\ltimes}$ in Section 5.4).

The remaining term operators $\triangleleft()$, $\triangleleft\mathtt{Inl}$, $\triangleleft\mathtt{Inr}$, $\triangleleft\mathtt{E}_{\mathrm{m}}$, $\triangleleft(,)$, $\triangleleft(\lambda x\,_{\mathrm{m}}\mapsto u)$ are all destination-filling primitives. They write a layer of value/constructor to the hole pointed by the destination operand, and return the potential new destinations that are created in the process (or unit if there is none).

5.4 Syntactic sugar for constructors and commonly used operations

sterm	::=		Syntactic sugar for terms
	$\mid$	`alloc` M	Evaluate to a fresh new ampar

|	$t \blacktriangleleft t'$	M	Fill destination with supplied term
|	$\mathtt{from}'_{\ltimes}\ t$	M	Extract left side of ampar when right side is unit
|	$^{s}\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\mapsto\ u$	M	Allocate function
|	$^{s}\mathtt{Inl}\ t$	M	Allocate left variant
|	$^{s}\mathtt{Inr}\ t$	M	Allocate right variant
|	$^{s}\mathsf{E}_{\mathsf{m}}\ t$	M	Allocate exponential
|	$^{s}(t_1\ ,\ t_2)$	M	Allocate product

$\mathtt{alloc}$	$\triangleq$	$_{\{1\}}\langle 1\ {}_{\wedge}\!\!\rightarrow\! 1\rangle$	$t \blacktriangleleft t'$	$\triangleq$	$t \blacktriangleleft\!\bullet (\mathtt{to}_{\ltimes}\ t')$
$\mathtt{from}'_{\ltimes}\ t$	$\triangleq$	$(\mathtt{from}_{\ltimes}\ (t \rhd \mathtt{map}\ \mathsf{un} \mapsto \mathsf{un}\ ; \mathsf{E}_{1\infty}\ ())) \rhd \mathtt{case}_{1\nu}$ $(\mathsf{st}\ ,\ \mathsf{ex}) \mapsto \mathsf{ex} \rhd \mathtt{case}_{1\nu}$ $\mathsf{E}_{1\infty}\ \mathsf{un} \mapsto \mathsf{un}\ ; \mathsf{st}$	$^{s}\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\mapsto\ u$	$\triangleq$	$\mathtt{from}'_{\ltimes}\ ($ $\mathtt{alloc} \rhd \mathtt{map}\ \mathsf{d} \mapsto$ $\mathsf{d} \blacktriangleleft (\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\mapsto\ u)$ $)$
$^{s}\mathtt{Inl}\ t$	$\triangleq$	$\mathtt{from}'_{\ltimes}\ ($ $\mathtt{alloc} \rhd \mathtt{map}\ \mathsf{d} \mapsto$ $\mathsf{d} \blacktriangleleft \mathtt{Inl} \blacktriangleleft t$ $)$	$^{s}\mathtt{Inr}\ t$	$\triangleq$	$\mathtt{from}'_{\ltimes}\ ($ $\mathtt{alloc} \rhd \mathtt{map}\ \mathsf{d} \mapsto$ $\mathsf{d} \blacktriangleleft \mathtt{Inr} \blacktriangleleft t$ $)$
$^{s}\mathsf{E}_{\mathsf{m}}\ t$	$\triangleq$	$\mathtt{from}'_{\ltimes}\ ($ $\mathtt{alloc} \rhd \mathtt{map}\ \mathsf{d} \mapsto$ $\mathsf{d} \blacktriangleleft \mathsf{E}_{\mathsf{m}} \blacktriangleleft t$ $)$	$^{s}(t_1\ ,\ t_2)$	$\triangleq$	$\mathtt{from}'_{\ltimes}\ ($ $\mathtt{alloc} \rhd \mathtt{map}\ \mathsf{d} \mapsto$ $(\mathsf{d} \blacktriangleleft (,)) \rhd \mathtt{case}_{1\nu}$ $(\mathsf{d}_1\ ,\ \mathsf{d}_2) \mapsto \mathsf{d}_1 \blacktriangleleft t_1\ ; \mathsf{d}_2 \blacktriangleleft t_2$ $)$

Table 1. Desugaring of syntactic sugar forms for terms

6 TYPE SYSTEM

6.1 Syntax for types, modes, and typing contexts

type, T, U	::=				Type
	|	1			Unit
	|	$T_1 \oplus T_2$			Sum
	|	$T_1 \otimes T_2$			Product
	|	$!_m T$			Exponential
	|	$U \ltimes T$			Ampar
	|	$T\ {}_m\!\!\to U$			Function
	|	$\lfloor_m T \rfloor$			Destination
mode, m, n	::=				Mode (Semiring)
	|	pa			Pair of a multiplicity and age
	|	💀			Error case (incompatible types, multiplicities, or ages)
mul, p	::=				Multiplicity (Semiring, first component of modality)
	|	1			Linear use
	|	ω			Non-linear use
age, a	::=				Age (Semiring, second component of modality)
	|	ν			Born now
	|	$\uparrow$			One scope older
	|	∞			Infinitely old / static
ctx, Γ, Δ, Θ	::=				Typing context
	|	$x :_m T$			Variable typing binding
	|	$h :_n T$			Hole typing binding
	|	$\to h :_m \lfloor_n T \rfloor$			Destination typing binding
	|	$m \cdot \Gamma$		M	Multiply the leftmost mode of each binding by m
	|	$\Gamma_1 + \Gamma_2$		M	Sum (incompatible bindings get tagged with 💀)
	|	$\Gamma_1 ,\ \Gamma_2$		M	Disjoint sum
	|	$\to^{-1}\Gamma$		M	Transforms dest bindings into a hole bindings
	|	$\to\Gamma$		M	Transforms hole bindings into dest bindings
	|	$\Gamma[H \pm h']$		M	Shift hole/dest names by h' if they belong to H

6.2 Typing of terms and values

$\boxed{\Gamma \Vdash v : \mathsf{T}}$ *(Typing judgment for values)*

$$\textsc{Ty-val-Hole}\quad \frac{}{\mathsf{h} :_{1\nu} \mathsf{T} \Vdash \mathsf{h} : \mathsf{T}}$$

$$\textsc{Ty-val-Dest}\quad \frac{}{\rightarrow\!\mathsf{h} :_{1\nu} \lfloor_{\mathsf{n}}\, \mathsf{T} \rfloor \Vdash \rightarrow\!\mathsf{h} : \lfloor_{\mathsf{n}}\, \mathsf{T} \rfloor}$$

$$\textsc{Ty-val-Unit}\quad \frac{}{\Vdash () : \mathsf{1}}$$

$$\textsc{Ty-val-Fun}\quad \frac{\Delta,\ \mathsf{x} :_{\mathsf{m}} \mathsf{T} \vdash u : \mathsf{U}}{\Delta \Vdash {}^{\mathsf{v}}\!\lambda \mathsf{x}\,{}_{\mathsf{m}}\!\mapsto u : \mathsf{T}\,{}_{\mathsf{m}}\!\!\to \mathsf{U}}$$

$$\textsc{Ty-val-Left}\quad \frac{\Gamma \Vdash v_1 : \mathsf{T}_1}{\Gamma \Vdash \mathtt{Inl}\ v_1 : \mathsf{T}_1 \oplus \mathsf{T}_2}$$

$$\textsc{Ty-val-Right}\quad \frac{\Gamma \Vdash v_2 : \mathsf{T}_2}{\Gamma \Vdash \mathtt{Inr}\ v_2 : \mathsf{T}_1 \oplus \mathsf{T}_2}$$

$$\textsc{Ty-val-Prod}\quad \frac{\Gamma_1 \Vdash v_1 : \mathsf{T}_1 \qquad \Gamma_2 \Vdash v_2 : \mathsf{T}_2}{\Gamma_1 + \Gamma_2 \Vdash (v_1\,,\ v_2) : \mathsf{T}_1 \otimes \mathsf{T}_2}$$

$$\textsc{Ty-val-Exp}\quad \frac{\Gamma \Vdash v' : \mathsf{T}}{\mathsf{n}\cdot\Gamma \Vdash \mathsf{E}_{\mathsf{n}}\ v' : !_{\mathsf{n}}\mathsf{T}}$$

$$\textsc{Ty-val-Ampar}\quad \frac{\mathtt{LinOnly}\ \Delta_3 \qquad \mathtt{FinAgeOnly}\ \Delta_3 \qquad 1\!\uparrow\!\cdot\Delta_1,\ \Delta_3 \Vdash v_1 : \mathsf{T} \qquad \Delta_2,\ (\rightarrow^{-1}\Delta_3) \Vdash v_2 : \mathsf{U}}{\Delta_1,\ \Delta_2 \Vdash_{\mathit{hvars}(\rightarrow^{-1}\Delta_3)} \langle v_2 \ {}_{\wedge}\ v_1 \rangle : \mathsf{U} \ltimes \mathsf{T}}$$

$\boxed{\Theta \vdash t : \mathsf{T}}$ *(Typing judgment for terms)*

$$\textsc{Ty-term-Val}\quad \frac{\mathtt{DisposableOnly}\ \Theta \qquad \Delta \Vdash v : \mathsf{T}}{\Theta,\ \Delta \vdash v : \mathsf{T}}$$

$$\textsc{Ty-term-Var}\quad \frac{\mathtt{DisposableOnly}\ \Theta \qquad 1\nu <: \mathsf{m}}{\Theta,\ \mathsf{x} :_{\mathsf{m}} \mathsf{T} \vdash \mathsf{x} : \mathsf{T}}$$

$$\textsc{Ty-term-App}\quad \frac{\Theta_1 \vdash t : \mathsf{T} \qquad \Theta_2 \vdash t' : \mathsf{T}\,{}_{\mathsf{m}}\!\!\to \mathsf{U}}{\mathsf{m}\cdot\Theta_1 + \Theta_2 \vdash t \triangleright t' : \mathsf{U}}$$

$$\textsc{Ty-term-PatU}\quad \frac{\Theta_1 \vdash t : \mathsf{1} \qquad \Theta_2 \vdash u : \mathsf{U}}{\Theta_1 + \Theta_2 \vdash t\ ;\ u : \mathsf{U}}$$

$$\textsc{Ty-term-PatS}\quad \frac{\Theta_1 \vdash t : \mathsf{T}_1 \oplus \mathsf{T}_2 \qquad \Theta_2,\ \mathsf{x}_1 :_{\mathsf{m}} \mathsf{T}_1 \vdash u_1 : \mathsf{U} \qquad \Theta_2,\ \mathsf{x}_2 :_{\mathsf{m}} \mathsf{T}_2 \vdash u_2 : \mathsf{U}}{\mathsf{m}\cdot\Theta_1 + \Theta_2 \vdash t \triangleright \mathtt{case}_{\mathsf{m}}\ \{\mathtt{Inl}\ \mathsf{x}_1 \mapsto u_1\,,\ \mathtt{Inr}\ \mathsf{x}_2 \mapsto u_2\} : \mathsf{U}}$$

$$\textsc{Ty-term-PatP}\quad \frac{\Theta_1 \vdash t : \mathsf{T}_1 \otimes \mathsf{T}_2 \qquad \Theta_2,\ \mathsf{x}_1 :_{\mathsf{m}} \mathsf{T}_1,\ \mathsf{x}_2 :_{\mathsf{m}} \mathsf{T}_2 \vdash u : \mathsf{U}}{\mathsf{m}\cdot\Theta_1 + \Theta_2 \vdash t \triangleright \mathtt{case}_{\mathsf{m}}\ (\mathsf{x}_1\,,\ \mathsf{x}_2) \mapsto u : \mathsf{U}}$$

$$\textsc{Ty-term-PatE}\quad \frac{\Theta_1 \vdash t : !_{\mathsf{n}}\mathsf{T} \qquad \Theta_2,\ \mathsf{x} :_{\mathsf{m}\cdot\mathsf{n}} \mathsf{T} \vdash u : \mathsf{U}}{\mathsf{m}\cdot\Theta_1 + \Theta_2 \vdash t \triangleright \mathtt{case}_{\mathsf{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathsf{x} \mapsto u : \mathsf{U}}$$

$$\textsc{Ty-term-Map}\quad \frac{\Theta_1 \vdash t : \mathsf{U} \ltimes \mathsf{T} \qquad 1\!\uparrow\!\cdot\Theta_2,\ \mathsf{x} :_{1\nu} \mathsf{T} \vdash t' : \mathsf{T}'}{\Theta_1 + \Theta_2 \vdash t \triangleright \mathtt{map}\ \mathsf{x} \mapsto t' : \mathsf{U} \ltimes \mathsf{T}'}$$

$$\textsc{Ty-term-ToA}\quad \frac{\Theta \vdash u : \mathsf{U}}{\Theta \vdash \mathtt{to}_{\ltimes}\ u : \mathsf{U} \ltimes \mathsf{1}}$$

$$\textsc{Ty-term-FromA}\quad \frac{\Theta \vdash t : \mathsf{U} \ltimes (!_{1\infty}\mathsf{T})}{\Theta \vdash \mathtt{from}_{\ltimes}\ t : \mathsf{U} \otimes (!_{1\infty}\mathsf{T})}$$

$$\textsc{Ty-term-FillU}\ \frac{\Theta \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{1} \rfloor}{\Theta \vdash t \triangleleft () : \mathsf{1}} \qquad \textsc{Ty-term-FillL}\ \frac{\Theta \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \oplus \mathsf{T}_2 \rfloor}{\Theta \vdash t \triangleleft \mathtt{Inl} : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \rfloor} \qquad \textsc{Ty-term-FillR}\ \frac{\Theta \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \oplus \mathsf{T}_2 \rfloor}{\Theta \vdash t \triangleleft \mathtt{Inr} : \lfloor_{\mathsf{n}}\, \mathsf{T}_2 \rfloor} \qquad \textsc{Ty-term-FillP}\ \frac{\Theta \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \otimes \mathsf{T}_2 \rfloor}{\Theta \vdash t \triangleleft (,) : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \rfloor \otimes \lfloor_{\mathsf{n}}\, \mathsf{T}_2 \rfloor}$$

$$\textsc{Ty-term-FillE}\ \frac{\Theta \vdash t : \lfloor_{\mathsf{n}}\, !_{\mathsf{n}'} \mathsf{T} \rfloor}{\Theta \vdash t \triangleleft \mathsf{E}_{\mathsf{n}'} : \lfloor_{\mathsf{n}' \cdot \mathsf{n}}\, \mathsf{T} \rfloor} \qquad \textsc{Ty-term-FillF}\ \frac{\Theta_1 \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{T}\ {}_{\mathsf{m}}\!\!\to \mathsf{U} \rfloor \qquad \Theta_2,\ \mathsf{x} :_{\mathsf{m}} \mathsf{T} \vdash u : \mathsf{U}}{\Theta_1 + (1\!\uparrow\!\cdot \mathsf{n})\cdot\Theta_2 \vdash t \triangleleft (\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\!\mapsto u) : \mathsf{1}} \qquad \textsc{Ty-term-FillComp}\ \frac{\Theta_1 \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{U} \rfloor \qquad \Theta_2 \vdash t' : \mathsf{U} \ltimes \mathsf{T}}{\Theta_1 + (1\!\uparrow\!\cdot \mathsf{n})\cdot\Theta_2 \vdash t \triangleleft\!\bullet\, t' : \mathsf{T}}$$

6.3 Derived typing rules for syntactic sugar forms

$\boxed{\Theta \ {}^{s}\!\vdash t : \mathsf{T}}$ *(Derived typing judgment for syntactic sugar forms)*

$$\textsc{Ty-sterm-Alloc}\ \frac{\mathtt{DisposableOnly}\ \Theta}{\Theta \ {}^{s}\!\vdash \mathtt{alloc} : \mathsf{T} \ltimes (\lfloor_{1\nu}\, \mathsf{T} \rfloor)} \qquad \textsc{Ty-sterm-FromA'}\ \frac{\Theta \vdash t : \mathsf{T} \ltimes \mathsf{1}}{\Theta \ {}^{s}\!\vdash \mathtt{from}'_{\ltimes}\ t : \mathsf{T}} \qquad \textsc{Ty-sterm-FillLeaf}\ \frac{\Theta_1 \vdash t : \lfloor_{\mathsf{n}}\, \mathsf{T} \rfloor \qquad \Theta_2 \vdash t' : \mathsf{T}}{\Theta_1 + (1\!\uparrow\!\cdot \mathsf{n})\cdot\Theta_2 \ {}^{s}\!\vdash t \triangleleft t' : \mathsf{1}}$$

$$\textsc{Ty-sterm-Fun}\ \frac{\Theta_2,\ \mathsf{x} :_{\mathsf{m}} \mathsf{T} \vdash u : \mathsf{U}}{\Theta_2 \ {}^{s}\!\vdash {}^{s}\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\!\mapsto u : \mathsf{T}\ {}_{\mathsf{m}}\!\!\to \mathsf{U}} \qquad \textsc{Ty-sterm-Left}\ \frac{\Theta_2 \vdash t : \mathsf{T}_1}{\Theta_2 \ {}^{s}\!\vdash {}^{s}\mathtt{Inl}\ t : \mathsf{T}_1 \oplus \mathsf{T}_2} \qquad \textsc{Ty-sterm-Right}\ \frac{\Theta_2 \vdash t : \mathsf{T}_2}{\Theta_2 \ {}^{s}\!\vdash {}^{s}\mathtt{Inr}\ t : \mathsf{T}_1 \oplus \mathsf{T}_2}$$

$$\textsc{Ty-sterm-Exp}\ \frac{\Theta_2 \vdash t : \mathsf{T}}{\mathsf{m}\cdot\Theta_2 \ {}^{s}\!\vdash {}^{s}\mathsf{E}_{\mathsf{m}}\ t : !_{\mathsf{m}} \mathsf{T}} \qquad \textsc{Ty-sterm-Prod}\ \frac{\Theta_{21} \vdash t_1 : \mathsf{T}_1 \qquad \Theta_{22} \vdash t_2 : \mathsf{T}_2}{\Theta_{21} + \Theta_{22} \ {}^{s}\!\vdash {}^{s}(t_1\,,\ t_2) : \mathsf{T}_1 \otimes \mathsf{T}_2}$$

7 EVALUATION CONTEXTS AND SEMANTICS

7.1 Evaluation contexts forms

$$
\begin{array}{lcll}
ectx,\ c & ::= & & \text{Evaluation context component} \\
 & | & \square \rhd t' & \\
 & | & v \rhd \square & \\
 & | & \square \,;\, u & \\
 & | & \square \rhd \mathtt{case}_{\mathsf{m}}\ \{\mathtt{Inl}\ \mathsf{x}_1 \mapsto u_1\,,\ \mathtt{Inr}\ \mathsf{x}_2 \mapsto u_2\} & \\
 & | & \square \rhd \mathtt{case}_{\mathsf{m}}\ (\mathsf{x}_1\,,\ \mathsf{x}_2) \mapsto u & \\
 & | & \square \rhd \mathtt{case}_{\mathsf{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathsf{x} \mapsto u & \\
 & | & \square \rhd \mathtt{map}\ \mathsf{x} \mapsto t' & \\
 & | & \mathtt{to}_{\ltimes}\ \square & \\
 & | & \mathtt{from}_{\ltimes}\ \square & \\
 & | & \square \triangleleft () & \\
 & | & \square \triangleleft \mathtt{Inl} & \\
 & | & \square \triangleleft \mathtt{Inr} & \\
 & | & \square \triangleleft \mathsf{E}_{\mathsf{m}} & \\
 & | & \square \triangleleft (,) & \\
 & | & \square \triangleleft (\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\!\mapsto u) & \\
 & | & \square \triangleleft\!\bullet\, t' & \\
 & | & v \triangleleft\!\bullet\, \square &
\end{array}
$$

	$\mid$	${}^{\mathrm{op}}_{\mathsf{H}}\langle v_2 \,{}_{\wedge} \square\rangle$	Open ampar, binding hole names in the next components
ectxs, C	$::=$		Evaluation context stack
	$\mid$	$\square$	Represent the empty stack / "identity" evaluation context
	$\mid$	$C \circ c$	Push c on top of C
	$\mid$	$C[\mathsf{h} :=_{\mathsf{H}} v]$ $\quad$ M	Fill h in C with value v (that may contain holes)

7.2 Typing of evaluation contexts and commands

$\boxed{\Delta \dashv C : \mathsf{T} \rightarrowtail \mathsf{U}_0}$ *(Typing judgment for evaluation contexts)*

$$\textsc{Ty-ectxs-Id}\quad \dfrac{}{\dashv \square : \mathsf{U}_0 \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-App-Foc1}\quad \dfrac{\mathsf{m}\cdot\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \rightarrowtail \mathsf{U}_0 \qquad \Delta_2 \vdash t' : \mathsf{T}\ {}_{\mathsf{m}}\!\!\to \mathsf{U}}{\Delta_1 \dashv C \circ (\square \rhd t') : \mathsf{T} \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-App-Foc2}\quad \dfrac{\mathsf{m}\cdot\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \rightarrowtail \mathsf{U}_0 \qquad \Delta_1 \vdash v : \mathsf{T}}{\Delta_2 \dashv C \circ (v \rhd \square) : (\mathsf{T}\ {}_{\mathsf{m}}\!\!\to \mathsf{U}) \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-PatU-Foc}\quad \dfrac{\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \rightarrowtail \mathsf{U}_0 \qquad \Delta_2 \vdash u : \mathsf{U}}{\Delta_1 \dashv C \circ (\square \,;\, u) : \mathsf{1} \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-PatS-Foc}\quad \dfrac{\mathsf{m}\cdot\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \rightarrowtail \mathsf{U}_0 \qquad \Delta_2,\ \mathsf{x}_1 :_{\mathsf{m}} \mathsf{T}_1 \vdash u_1 : \mathsf{U} \qquad \Delta_2,\ \mathsf{x}_2 :_{\mathsf{m}} \mathsf{T}_2 \vdash u_2 : \mathsf{U}}{\Delta_1 \dashv C \circ (\square \rhd \mathtt{case}_{\mathsf{m}}\, \{\mathtt{Inl}\ \mathsf{x}_1 \mapsto u_1\,,\ \mathtt{Inr}\ \mathsf{x}_2 \mapsto u_2\}) : (\mathsf{T}_1 \oplus \mathsf{T}_2) \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-PatP-Foc}\quad \dfrac{\mathsf{m}\cdot\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \rightarrowtail \mathsf{U}_0 \qquad \Delta_2,\ \mathsf{x}_1 :_{\mathsf{m}} \mathsf{T}_1,\ \mathsf{x}_2 :_{\mathsf{m}} \mathsf{T}_2 \vdash u : \mathsf{U}}{\Delta_1 \dashv C \circ (\square \rhd \mathtt{case}_{\mathsf{m}}\, (\mathsf{x}_1\,,\ \mathsf{x}_2) \mapsto u) : (\mathsf{T}_1 \otimes \mathsf{T}_2) \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-PatE-Foc}\quad \dfrac{\mathsf{m}\cdot\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \rightarrowtail \mathsf{U}_0 \qquad \Delta_2,\ \mathsf{x} :_{\mathsf{m}\cdot\mathsf{m}'} \mathsf{T} \vdash u : \mathsf{U}}{\Delta_1 \dashv C \circ (\square \rhd \mathtt{case}_{\mathsf{m}}\, \mathsf{E}_{\mathsf{m}'}\ \mathsf{x} \mapsto u) : !_{\mathsf{m}'}\mathsf{T} \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-Map-Foc}\quad \dfrac{\Delta_1,\ \Delta_2 \dashv C : \mathsf{U} \ltimes \mathsf{T}' \rightarrowtail \mathsf{U}_0 \qquad 1{\uparrow}\cdot\Delta_2,\ \mathsf{x} :_{1\nu} \mathsf{T} \vdash t' : \mathsf{T}'}{\Delta_1 \dashv C \circ (\square \rhd \mathtt{map}\ \mathsf{x} \mapsto t') : (\mathsf{U} \ltimes \mathsf{T}) \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-ToA-Foc}\quad \dfrac{\Delta \dashv C : (\mathsf{U} \ltimes \mathsf{1}) \rightarrowtail \mathsf{U}_0}{\Delta \dashv C \circ (\mathtt{to}_{\ltimes}\ \square) : \mathsf{U} \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-FromA-Foc}\quad \dfrac{\Delta \dashv C : (\mathsf{U} \otimes (!_{1\infty}\mathsf{T})) \rightarrowtail \mathsf{U}_0}{\Delta \dashv C \circ (\mathtt{from}_{\ltimes}\ \square) : (\mathsf{U} \ltimes (!_{1\infty}\mathsf{T})) \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-FillU-Foc}\quad \dfrac{\Delta \dashv C : \mathsf{1} \rightarrowtail \mathsf{U}_0}{\Delta \dashv C \circ (\square \blacktriangleleft ()) : \lfloor_{\mathsf{n}}\, \mathsf{1} \rfloor \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-FillL-Foc}\quad \dfrac{\Delta \dashv C : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \rfloor \rightarrowtail \mathsf{U}_0}{\Delta \dashv C \circ (\square \blacktriangleleft \mathtt{Inl}) : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \oplus \mathsf{T}_2 \rfloor \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-FillR-Foc}\quad \dfrac{\Delta \dashv C : \lfloor_{\mathsf{n}}\, \mathsf{T}_2 \rfloor \rightarrowtail \mathsf{U}_0}{\Delta \dashv C \circ (\square \blacktriangleleft \mathtt{Inr}) : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \oplus \mathsf{T}_2 \rfloor \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-FillP-Foc}\quad \dfrac{\Delta \dashv C : (\lfloor_{\mathsf{n}}\, \mathsf{T}_1 \rfloor \otimes \lfloor_{\mathsf{n}}\, \mathsf{T}_2 \rfloor) \rightarrowtail \mathsf{U}_0}{\Delta \dashv C \circ (\square \blacktriangleleft (,)) : \lfloor_{\mathsf{n}}\, \mathsf{T}_1 \otimes \mathsf{T}_2 \rfloor \rightarrowtail \mathsf{U}_0}$$

$$\textsc{Ty-ectxs-FillE-Foc}\quad \dfrac{\Delta \dashv C : \lfloor_{m\cdot n}\, T \rfloor \rightarrowtail U_0}{\Delta \dashv C \circ (\square \triangleleft \mathsf{E}_m) : \lfloor_{n}\, !_m T \rfloor \rightarrowtail U_0}$$

$$\textsc{Ty-ectxs-FillF-Foc}\quad \dfrac{\Delta_1,\ (1\uparrow\cdot n)\cdot\Delta_2 \dashv C : 1 \rightarrowtail U_0 \qquad \Delta_2,\ x :_m T \vdash u : U}{\Delta_1 \dashv C \circ (\square \triangleleft (\lambda x\ {}_m\!\mapsto u)) : \lfloor_{n}\, T\ {}_m\!\!\to U \rfloor \rightarrowtail U_0}$$

$$\textsc{Ty-ectxs-FillComp-Foc1}\quad \dfrac{\Delta_1,\ (1\uparrow\cdot n)\cdot\Delta_2 \dashv C : T \rightarrowtail U_0 \qquad \Delta_2 \vdash t' : U \ltimes T}{\Delta_1 \dashv C \circ (\square \triangleleft\!\bullet\, t') : \lfloor_{n}\, U \rfloor \rightarrowtail U_0}$$

$$\textsc{Ty-ectxs-FillComp-Foc2}\quad \dfrac{\Delta_1,\ (1\uparrow\cdot n)\cdot\Delta_2 \dashv C : T \rightarrowtail U_0 \qquad \Delta_1 \vdash v : \lfloor_{n}\, U \rfloor}{\Delta_2 \dashv C \circ (v \triangleleft\!\bullet\, \square) : U \ltimes T \rightarrowtail U_0}$$

$$\textsc{Ty-ectxs-OpenAmpar-Foc}\quad \dfrac{\begin{array}{c} hvars(C)\ \#\#\ hvars(\to^{-1}\Delta_3) \\ \mathtt{LinOnly}\ \Delta_3 \\ \mathtt{FinAgeOnly}\ \Delta_3 \\ \Delta_1,\ \Delta_2 \dashv C : (U \ltimes T') \rightarrowtail U_0 \\ \Delta_2,\ \to^{-1}\Delta_3 \Vdash v_2 : U \end{array}}{1\uparrow\cdot\Delta_1,\ \Delta_3 \dashv C \circ ({}^{\mathrm{op}}_{hvars(\to^{-1}\Delta_3)}\langle v_2\ {}_{\wedge}\, \square \rangle) : T' \rightarrowtail U_0}$$

$\boxed{\vdash C[t] : T}$ *(Typing judgment for commands)*

$$\textsc{Ty-cmd}\quad \dfrac{\Delta \dashv C : T \rightarrowtail U_0 \qquad \Delta \vdash t : T}{\vdash C[t] : U_0}$$

7.3 Small-step semantics

$\boxed{C[t] \longrightarrow C'[t']}$ *(Small-step evaluation of commands)*

$$\textsc{Sem-App-Foc1}\quad \dfrac{\mathtt{NotVal}\ t}{C[t \triangleright t'] \longrightarrow (C \circ (\square \triangleright t'))[t]}$$

$$\textsc{Sem-App-Unfoc1}\quad \dfrac{}{(C \circ (\square \triangleright t'))[v] \longrightarrow C[v \triangleright t']}$$

$$\textsc{Sem-App-Foc2}\quad \dfrac{\mathtt{NotVal}\ t'}{C[v \triangleright t'] \longrightarrow (C \circ (v \triangleright \square))[t']}$$

$$\textsc{Sem-App-Unfoc2}\quad \dfrac{}{(C \circ (v \triangleright \square))[v'] \longrightarrow C[v \triangleright v']}$$

$$\textsc{Sem-App-Red}\quad \dfrac{}{C[v \triangleright ({}^{v}\!\lambda x\ {}_m\!\mapsto u)] \longrightarrow C[u[x := v]]}$$

$$\textsc{Sem-PatU-Foc}\quad \dfrac{\mathtt{NotVal}\ t}{C[t\ ;\ u] \longrightarrow (C \circ (\square\ ;\ u))[t]}$$

$$\textsc{Sem-PatU-Unfoc}\quad \dfrac{}{(C \circ (\square\ ;\ u))[v] \longrightarrow C[v\ ;\ u]}$$

$$\textsc{Sem-PatU-Red}\quad \dfrac{}{C[()\ ;\ u] \longrightarrow C[u]}$$

$$\textsc{Sem-PatS-Foc}\quad \dfrac{\mathtt{NotVal}\ t}{C[t \triangleright \mathtt{case}_m\,\{\mathtt{Inl}\, x_1 \mapsto u_1,\ \mathtt{Inr}\, x_2 \mapsto u_2\}] \longrightarrow (C \circ (\square \triangleright \mathtt{case}_m\,\{\mathtt{Inl}\, x_1 \mapsto u_1,\ \mathtt{Inr}\, x_2 \mapsto u_2\}))[t]}$$

$$\textsc{Sem-PatS-Unfoc}\quad \frac{}{(C \circ (\square \rhd \mathtt{case}_{\mathtt{m}}\ \{\mathtt{Inl}\ \mathtt{x}_1 \mapsto u_1\ ,\ \mathtt{Inr}\ \mathtt{x}_2 \mapsto u_2\}))[v] \longrightarrow C[v \rhd \mathtt{case}_{\mathtt{m}}\ \{\mathtt{Inl}\ \mathtt{x}_1 \mapsto u_1\ ,\ \mathtt{Inr}\ \mathtt{x}_2 \mapsto u_2\}]}$$

$$\textsc{Sem-PatL-Red}\quad \frac{}{C[(\mathtt{Inl}\ v_1) \rhd \mathtt{case}_{\mathtt{m}}\ \{\mathtt{Inl}\ \mathtt{x}_1 \mapsto u_1\ ,\ \mathtt{Inr}\ \mathtt{x}_2 \mapsto u_2\}] \longrightarrow C[u_1[\mathtt{x}_1 \coloneqq v_1]]}$$

$$\textsc{Sem-PatR-Red}\quad \frac{}{C[(\mathtt{Inr}\ v_2) \rhd \mathtt{case}_{\mathtt{m}}\ \{\mathtt{Inl}\ \mathtt{x}_1 \mapsto u_1\ ,\ \mathtt{Inr}\ \mathtt{x}_2 \mapsto u_2\}] \longrightarrow C[u_2[\mathtt{x}_2 \coloneqq v_2]]}$$

$$\textsc{Sem-PatP-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \rhd \mathtt{case}_{\mathtt{m}}\ (\mathtt{x}_1\ ,\ \mathtt{x}_2) \mapsto u] \longrightarrow (C \circ (\square \rhd \mathtt{case}_{\mathtt{m}}\ (\mathtt{x}_1\ ,\ \mathtt{x}_2) \mapsto u))[t]}$$

$$\textsc{Sem-PatP-Unfoc}\quad \frac{}{(C \circ (\square \rhd \mathtt{case}_{\mathtt{m}}\ (\mathtt{x}_1\ ,\ \mathtt{x}_2) \mapsto u))[v] \longrightarrow C[v \rhd \mathtt{case}_{\mathtt{m}}\ (\mathtt{x}_1\ ,\ \mathtt{x}_2) \mapsto u]}$$

$$\textsc{Sem-PatP-Red}\quad \frac{}{C[(v_1\ ,\ v_2) \rhd \mathtt{case}_{\mathtt{m}}\ (\mathtt{x}_1\ ,\ \mathtt{x}_2) \mapsto u] \longrightarrow C[u[\mathtt{x}_1 \coloneqq v_1][\mathtt{x}_2 \coloneqq v_2]]}$$

$$\textsc{Sem-PatE-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \rhd \mathtt{case}_{\mathtt{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathtt{x} \mapsto u] \longrightarrow (C \circ (\square \rhd \mathtt{case}_{\mathtt{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathtt{x} \mapsto u))[t]}$$

$$\textsc{Sem-PatE-Unfoc}\quad \frac{}{(C \circ (\square \rhd \mathtt{case}_{\mathtt{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathtt{x} \mapsto u))[v] \longrightarrow C[v \rhd \mathtt{case}_{\mathtt{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathtt{x} \mapsto u]}$$

$$\textsc{Sem-PatE-Red}\quad \frac{}{C[\mathsf{E}_{\mathsf{n}}\ v' \rhd \mathtt{case}_{\mathtt{m}}\ \mathsf{E}_{\mathsf{n}}\ \mathtt{x} \mapsto u] \longrightarrow C[u[\mathtt{x} \coloneqq v']]}$$

$$\textsc{Sem-Map-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \rhd \mathtt{map}\ \mathtt{x} \mapsto t'] \longrightarrow (C \circ (\square \rhd \mathtt{map}\ \mathtt{x} \mapsto t'))[t]}$$

$$\textsc{Sem-Map-Unfoc}\quad \frac{}{(C \circ (\square \rhd \mathtt{map}\ \mathtt{x} \mapsto t'))[v] \longrightarrow C[v \rhd \mathtt{map}\ \mathtt{x} \mapsto t']}$$

$$\textsc{Sem-Map-Red-OpenAmpar-Foc}\quad \frac{\mathsf{h}' \ = \ \mathit{max}(\mathit{hvars}(C)){+}1}{C[{}_{\mathsf{H}}\langle v_2\ {}_{\wedge}\ v_1\rangle \rhd \mathtt{map}\ \mathtt{x} \mapsto t'] \longrightarrow (C \circ ({}^{\mathrm{op}}_{\mathsf{H}\pm\mathsf{h}'}\langle v_2[\mathsf{H}\pm\mathsf{h}']\ {}_{\wedge}\square\rangle))[t'[\mathtt{x} \coloneqq v_1[\mathsf{H}\pm\mathsf{h}']]]}$$

$$\textsc{Sem-OpenAmpar-Unfoc}\quad \frac{}{(C \circ^{\mathrm{op}}_{\mathsf{H}} \langle v_2\ {}_{\wedge}\square\rangle)[v_1] \longrightarrow C[{}_{\mathsf{H}}\langle v_2\ {}_{\wedge}\ v_1\rangle]}$$

$$\textsc{Sem-ToA-Foc}\quad \frac{\mathtt{NotVal}\ \ u}{C[\mathtt{to}_{\ltimes}\ u] \longrightarrow (C \circ (\mathtt{to}_{\ltimes}\ \square))[u]}$$

$$\textsc{Sem-ToA-Unfoc}\quad \frac{}{(C \circ (\mathtt{to}_{\ltimes}\ \square))[v_2] \longrightarrow C[\mathtt{to}_{\ltimes}\ v_2]}$$

$$\textsc{Sem-ToA-Red}\quad \frac{}{C[\mathtt{to}_{\ltimes}\ v_2] \longrightarrow C[_{\{\,\}}\langle v_2\ {}_{\wedge}()\rangle]}$$

$$\textsc{Sem-FromA-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[\mathtt{from}_{\ltimes}\ t] \longrightarrow (C \circ (\mathtt{from}_{\ltimes}\ \square))[t]}$$

$$\textsc{Sem-FromA-Unfoc}\quad \frac{}{(C \circ (\mathtt{from}_{\ltimes}\ \square))[v] \longrightarrow C[\mathtt{from}_{\ltimes}\ v]}$$

$$\textsc{Sem-FromA-Red}\quad \frac{}{C[\mathtt{from}_{\ltimes}\ _{\{\,\}}\langle v_2\ {}_{\wedge}\mathsf{E}_{1\infty}\ v_1\rangle] \longrightarrow C[(v_2\,,\ \mathsf{E}_{1\infty}\ v_1)]}$$

$$\textsc{Sem-FillU-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \triangleleft ()] \longrightarrow (C \circ (\square \triangleleft ()))[t]}$$

$$\textsc{Sem-FillU-Unfoc}\quad \frac{}{(C \circ (\square \triangleleft ()))[v] \longrightarrow C[v \triangleleft ()]}$$

$$\textsc{Sem-FillU-Red}\quad \frac{}{C[\rightarrow\mathsf{h} \triangleleft ()] \longrightarrow C[\mathsf{h} \coloneqq_{\{\,\}}\ ()][()]}$$

$$\textsc{Sem-FillL-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \triangleleft \mathtt{Inl}] \longrightarrow (C \circ (\square \triangleleft \mathtt{Inl}))[t]}$$

$$\textsc{Sem-FillL-Unfoc}\quad \frac{}{(C \circ (\square \triangleleft \mathtt{Inl}))[v] \longrightarrow C[v \triangleleft \mathtt{Inl}]}$$

$$\textsc{Sem-FillL-Red}\quad \frac{\mathsf{h}' \ =\ \mathit{max}(\mathit{hvars}(C) \cup \{\mathsf{h}\})+1}{C[\rightarrow\mathsf{h} \triangleleft \mathtt{Inl}] \longrightarrow C[\mathsf{h} \coloneqq_{\{\mathsf{h}'+1\}}\ \mathtt{Inl}\ (\mathsf{h}'+1)][\rightarrow(\mathsf{h}'+1)]}$$

$$\textsc{Sem-FillR-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \triangleleft \mathtt{Inr}] \longrightarrow (C \circ (\square \triangleleft \mathtt{Inr}))[t]}$$

$$\textsc{Sem-FillR-Unfoc}\quad \frac{}{(C \circ (\square \triangleleft \mathtt{Inr}))[v] \longrightarrow C[v \triangleleft \mathtt{Inr}]}$$

$$\textsc{Sem-FillR-Red}\quad \frac{\mathsf{h}' \ =\ \mathit{max}(\mathit{hvars}(C) \cup \{\mathsf{h}\})+1}{C[\rightarrow\mathsf{h} \triangleleft \mathtt{Inr}] \longrightarrow C[\mathsf{h} \coloneqq_{\{\mathsf{h}'+1\}}\ \mathtt{Inr}\ (\mathsf{h}'+1)][\rightarrow(\mathsf{h}'+1)]}$$

$$\textsc{Sem-FillE-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \triangleleft \mathsf{E}_{\mathsf{m}}] \longrightarrow (C \circ (\square \triangleleft \mathsf{E}_{\mathsf{m}}))[t]}$$

$$\textsc{Sem-FillE-Unfoc}\quad \frac{}{(C \circ (\square \triangleleft \mathsf{E}_{\mathsf{m}}))[v] \longrightarrow C[v \triangleleft \mathsf{E}_{\mathsf{m}}]}$$

$$\textsc{Sem-FillE-Red}\quad \frac{\mathsf{h}' \ =\ \mathit{max}(\mathit{hvars}(C) \cup \{\mathsf{h}\})+1}{C[\rightarrow\mathsf{h} \triangleleft \mathsf{E}_{\mathsf{m}}] \longrightarrow C[\mathsf{h} \coloneqq_{\{\mathsf{h}'+1\}}\ \mathsf{E}_{\mathsf{m}}\ (\mathsf{h}'+1)][\rightarrow(\mathsf{h}'+1)]}$$

$$\textsc{Sem-FillP-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \triangleleft (,)] \longrightarrow (C \circ (\square \triangleleft (,)))[t]}$$

$$\textsc{Sem-FillP-Unfoc}\quad \frac{}{(C \circ (\square \triangleleft (,)))[v] \longrightarrow C[v \triangleleft (,)]}$$

$$\textsc{Sem-FillP-Red}\quad \frac{\mathsf{h}' \ =\ \mathit{max}(\mathit{hvars}(C) \cup \{\mathsf{h}\})+1}{C[\rightarrow\mathsf{h} \triangleleft (,)] \longrightarrow C[\mathsf{h} \coloneqq_{\{\mathsf{h}'+1,\mathsf{h}'+2\}}\ ((\mathsf{h}'+1)\,,\ (\mathsf{h}'+2))][(\rightarrow(\mathsf{h}'+1)\,,\ \rightarrow(\mathsf{h}'+2))]}$$

$$\textsc{Sem-FillF-Foc}\quad \frac{\mathtt{NotVal}\ \ t}{C[t \triangleleft (\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\mapsto\ u)] \longrightarrow (C \circ (\square \triangleleft (\lambda \mathsf{x}\ {}_{\mathsf{m}}\!\mapsto\ u)))[t]}$$

$$\frac{}{(C \circ (\square \triangleleft (\lambda \mathsf{x}\,_{\mathsf{m}}\!\mapsto\; u)))[v] \longrightarrow C[v \triangleleft (\lambda \mathsf{x}\,_{\mathsf{m}}\!\mapsto\; u)]}\;\textsc{Sem-FillF-Unfoc}$$

$$\frac{}{C[\rightarrow\!\mathsf{h} \triangleleft (\lambda \mathsf{x}\,_{\mathsf{m}}\!\mapsto\; u)] \longrightarrow C[\mathsf{h} \coloneqq_{\{\}}\; {}^{\mathrm{v}}\!\lambda \mathsf{x}\,_{\mathsf{m}}\!\mapsto\; u][()]}\;\textsc{Sem-FillF-Red}$$

$$\frac{\mathtt{NotVal}\;\; t}{C[t \triangleleft\!\bullet\, t'] \longrightarrow (C \circ (\square \triangleleft\!\bullet\, t'))[t]}\;\textsc{Sem-FillComp-Foc1}$$

$$\frac{}{(C \circ (\square \triangleleft\!\bullet\, t'))[v] \longrightarrow C[v \triangleleft\!\bullet\, t']}\;\textsc{Sem-FillComp-Unfoc1}$$

$$\frac{\mathtt{NotVal}\;\; t'}{C[v \triangleleft\!\bullet\, t'] \longrightarrow (C \circ (v \triangleleft\!\bullet\, \square))[t']}\;\textsc{Sem-FillComp-Foc2}$$

$$\frac{}{(C \circ (v \triangleleft\!\bullet\, \square))[v'] \longrightarrow C[v \triangleleft\!\bullet\, v']}\;\textsc{Sem-FillComp-Unfoc2}$$

$$\frac{\mathsf{h}' \;=\; \mathit{max}(\mathit{hvars}(C) \cup \{\mathsf{h}\}) + 1}{C[\rightarrow\!\mathsf{h} \triangleleft\!\bullet\, {}_{\mathsf{H}}\langle v_2 \,{}_{\wedge}\, v_1\rangle] \longrightarrow C[\mathsf{h} \coloneqq_{(\mathsf{H} \pm \mathsf{h}')}\; v_2[\mathsf{H} \pm \mathsf{h}']][v_1[\mathsf{H} \pm \mathsf{h}']]}\;\textsc{Sem-FillComp-Red}$$

8 PROOF OF TYPE SAFETY USING COQ PROOF ASSISTANT

- Not particularly elegant. Max number of goals observed 232 (solved by a single call to the `congruence` tactic). When you have a computer, brute force is a viable strategy. (in particular, no semiring formalisation, it was quicker to do directly)
- Rules generated by ott, same as in the article (up to some notational difference). Contexts are not generated purely by syntax, and are interpreted in a semantic domain (finite functions).
- Reasoning on closed terms avoids almost all complications on binder manipulation. Makes proofs tractable.
- Finite functions: making a custom library was less headache than using existing libraries (including `MMap`). Existing libraries don't provide some of the tools that we needed, but the most important factor ended up being the need for a modicum of dependency between key and value. There wasn't really that out there. Backed by actual functions for simplicity; cost: equality is complicated.
- Most of the proofs done by author with very little prior experience to Coq.
- Did proofs in Coq because context manipulations are tricky.
- Context sum made total by adding an extra invalid *mode* (rather than an extra context). It seems to be much simpler this way.
- It might be a good idea to provide statistics on the number of lemmas and size of Coq codebase.
- (possibly) renaming as permutation, inspired by nominal sets, make more lemmas don't require a condition (but some lemmas that wouldn't in a straight renaming do in exchange).
- (possibly) methodology: assume a lot of lemmas, prove main theorem, prove assumptions, some wrong, fix. A number of wrong lemma initially assumed, but replacing them by correct variant was always easy to fix in proofs.
- Axioms that we use and why (in particular setoid equality not very natural with ott-generated typing rules).
- Talk about the use and benefits of Copilot.

9 IMPLEMENTATION OF DESTINATION CALCULUS USING IN-PLACE MEMORY MUTATIONS

What needs to be changed (e.g. linear alloc)

10 RELATED WORK

11 CONCLUSION AND FUTURE WORK

REFERENCES

[1] Thomas Bagrel. 2024. Destination-passing style programming: a Haskell implementation. https://inria.hal.science/hal-04406360

[2] Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. 2018. Linear Haskell: practical linearity in a higher-order polymorphic language. *Proceedings of the ACM on Programming Languages* 2, POPL (Jan. 2018), 1–29. https://doi.org/10.1145/3158093 arXiv:1710.09756 [cs].

[3] Yasuhiko Minamide. 1998. A functional representation of data structures with a hole. In *Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '98)*. Association for Computing Machinery, New York, NY, USA, 75–84. https://doi.org/10.1145/268946.268953