Linear Effects, Exceptions, and Resource Safety

Monday 13th October, 2025

Programming with resources

Resources are values that are difficult to copy or to erase, e.g. file handles.

```
let f = open_file("record.txt");
append(f, "starting tasks");
run_tasks(f);
close_file(f);
```

May introduce bugs such as: not closing the file, closing it twice, using it after closing it, ...

Linearity

We model resources with *linearity*: values will be consumed exactly once.

Ordered fragment: values cannot be exchanged. The most recent resource will be consumed first.

Minimal setting: e.g. no temporary access to resources (mutable borrowing).

We now need effectful operations to acquire and to free abstract resources.

The allocation monad

We define (using linear logic notation, e.g. 1 neutral for $\otimes$):

- An atomic type R.
- The type of lists of $R \stackrel{\text{def}}{=} [R]$.
- The linear state monad $TA \stackrel{\text{def}}{=} [R] \multimap (A \otimes [R])$.
- **new** : $1 \to T(R \oplus 1) \simeq [R] \to (R \oplus 1) \otimes [R]$
  $\stackrel{\text{def}}{=} \mathrm{case}\{[] \mapsto (\iota_2 \star, []) \mid h :: t \mapsto (\iota_1 h, t)\}$.
- **delete** : $R \to T1 \simeq R \otimes [R] \to [R] \stackrel{\text{def}}{=} \lambda(h, t).h :: t$.
- Its left strength $A \otimes TB \to T(A \otimes B) \simeq A \otimes TB \otimes [R] \to A \otimes B \otimes [R]$
  $\stackrel{\text{def}}{=} \lambda(a, tb, l_1).\, \mathrm{let}(b, l_2) = tb\ l_1\ \mathrm{in}(a, b, l_2)$.

Linear call-by-push-value - Grammar

Call-by-push-value: classify expressions by polarities $\varepsilon \stackrel{\text{def}}{=} + \mid -$ which determine their evaluation order ($+ \rightsquigarrow$ call-by-value, $- \rightsquigarrow$ call-by-name).

Grammar gives untyped expressions and distinguish values as substituable expressions : hence, negative expressions are always values (following call-by-name).

$$
\begin{array}{lll}
\text{Expressions} & t, u ::= & v \mid (\text{let } x^+ = t \text{ in } u)^+ \mid (\text{let } x^- = v \text{ in } u)^+ \mid \delta(v, (x, y).t)^+ \mid \delta(v, ().t)^+ \mid \\
& & \delta(v, x.t, y.u)^+ \mid (vw)^+ \mid (\pi_1 v)^+ \mid (\pi_2 v)^+ \\
\text{Values} & v, w ::= & (\text{let } x^+ = t \text{ in } v)^- \mid (\text{let } x^- = v \text{ in } w)^- \mid \delta(v, (x, y).w)^- \mid \delta(v, ().w)^- \mid \\
& & \delta(v, x.w, y.w')^- \mid (vw)^- \mid (\pi_1 v)^- \mid (\pi_2 v)^- \mid \\
& & x \mid \textbf{new} \mid \textbf{delete} \mid (v, w) \mid () \mid \iota_1 v \mid \iota_2 v \mid \lambda x.t \mid \langle t, u \rangle
\end{array}
$$

Linear call-by-push-value - Typing rules (1)

$$\text{Types } A, B : \begin{cases} \text{Positive types} & P, Q ::= R \mid 1 \mid A \otimes B \mid A \oplus B \\ \text{Negative types} & N, M ::= B \multimap A \mid A \,\&\, B \end{cases}$$

Type polarities $\varpi(P) \stackrel{\text{def}}{=} +, \varpi(N) \stackrel{\text{def}}{=} -$.

Follows *IMALL* rules with value restriction. T remains implicit. $\Sigma(\Gamma, \Gamma')$ is the set of variables permutations and renamings.

$$\frac{}{x : A \vdash x : A}\ \text{var} \qquad \frac{\Gamma \vdash t : A \qquad \sigma \in \Sigma(\Gamma, \Gamma')}{\Gamma' \vdash t[\sigma] : A}\ \text{struct}$$

$$\frac{}{\vdash \mathbf{new} : (R \oplus 1) \multimap 1}\ \text{new} \qquad \frac{}{\vdash \mathbf{delete} : 1 \multimap R}\ \text{delete}$$

$$\frac{\Delta \vdash t : A_{\varepsilon} \qquad \Gamma, x : A \vdash u : B_{\varepsilon'}}{\Gamma, \Delta \vdash (\text{let } x^{\varepsilon} = t \text{ in } u)^{\varepsilon'} : B}\ \text{let}$$

Linear call-by-push-value - Typing rules (2)

$$\frac{\Gamma \vdash v : A \qquad \Delta \vdash w : B}{\Gamma, \Delta \vdash (v, w) : A \otimes B}\ \otimes_i \qquad \frac{\Delta \vdash v : A \otimes B \qquad \Gamma, x : A, y : B, \Gamma' \vdash t : C_\varepsilon}{\Gamma, \Delta, \Gamma' \vdash \delta(v, (x, y).t)^\varepsilon : C}\ \otimes_e$$

$$\frac{}{\vdash () : 1}\ 1_i \qquad \frac{\Delta \vdash v : 1 \qquad \Gamma, \Gamma' \vdash t : A_\varepsilon}{\Gamma, \Delta, \Gamma' \vdash \delta(v, ().t)^\varepsilon : A}\ 1_e$$

$$\frac{\Gamma \vdash v : A}{\Gamma \vdash \iota_1 v : A \oplus B}\ \oplus_{i1} \qquad \frac{\Gamma \vdash v : B}{\Gamma \vdash \iota_2 v : A \oplus B}\ \oplus_{i2}$$

$$\frac{\Delta \vdash v : A \oplus B \qquad \Gamma, x : A, \Gamma' \vdash t : C_\varepsilon \qquad \Gamma, y : B, \Gamma' \vdash u : C}{\Gamma, \Delta, \Gamma' \vdash \delta(v, x.t, y.u)^\varepsilon : C}\ \oplus_e$$

$$\frac{x : A, \Gamma \vdash t : B}{\Gamma \vdash \lambda x.t : B \multimap A}\ \multimap_i \qquad \frac{\Gamma \vdash w : A \qquad \Delta \vdash v : B_\varepsilon \multimap A}{\Gamma, \Delta \vdash (vw)^\varepsilon : B}\ \multimap_e$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash \langle t, u \rangle : A \,\&\, B}\ \&_i \qquad \frac{\Gamma \vdash v : A_\varepsilon \,\&\, B}{\Gamma \vdash (\pi_1 v)^\varepsilon : A}\ \&_{e1} \qquad \frac{\Gamma \vdash v : A \,\&\, B_\varepsilon}{\Gamma \vdash (\pi_2 v)^\varepsilon : B}\ \&_{e2}$$

Linear call-by-push-value - example

We call this language $\mathcal{L}$, its ordered fragment $\mathcal{O}$ (without permutations in $\Sigma(\Gamma, \Gamma')$). Functions $\circ\!\!-$ bind the *leftmost* variable, let expressions bind the *rightmost* variable (like the function $\multimap$ in T).

Example in $\mathcal{O}$ that allocates then frees two resources:

$$\vdash \mathrm{let}\ a = \mathbf{new}() \ \mathrm{in}\ \delta(a, r.\, \mathrm{let}\ b = \mathbf{new}() \ \mathrm{in}\ \delta(b, s.\, \mathbf{delete}\ s; \mathbf{delete}\ r, i.i; \mathbf{delete}\ r), i.i) : 1$$

Linearity forces us to deal explicitly with the error case $(i; \mathbf{delete}\ r)$.

Central types

We distinguish *central types* W: semantically, have (coherent, natural) isomorphisms $\mathrm{swap} : A \otimes W \to W \otimes A$.

Syntactically, swap is definable in $\boldsymbol{O}$ for types in $W ::= 1 \mid W \oplus W' \mid W \otimes W'$. They do not contain resources.

Used later for exceptions and to state resource-safety properties.

Operational semantics - abstract machine

Direct small-step operational semantics with an abstract machine storing the ambient list of resources:

Given a list l, run $\vdash t : A$ in $\langle t \mid \star \mid l \rangle$.

$$\begin{aligned} \text{Stacks } s &::= \star \mid v^{\varepsilon} \cdot s \mid \pi_i^{\varepsilon} \cdot s \mid (x^{+}.u)^{\varepsilon} \cdot s \\ \text{Lists } l &::= [] \mid r_{n \in \mathbb{N}} :: l \\ \text{Commands } c &::= \langle t \mid s \mid l \rangle^{\varepsilon} \end{aligned}$$

Operational semantics - reduction steps

$$\langle (\text{let } x^- = v \text{ in } t)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle t[v/x] \mid s \mid l \rangle^\varepsilon$$

$$\langle (\text{let } x^+ = t \text{ in } u)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle t \mid (x^+.u)^\varepsilon \cdot s \mid l \rangle^+ \qquad \langle v \mid (x^+.t)^\varepsilon \cdot s \mid l \rangle^+ \rightsquigarrow \langle t[v/x] \mid s \mid l \rangle^\varepsilon$$

$$\langle (vw)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle v \mid w^\varepsilon \cdot s \mid l \rangle^- \qquad \langle \lambda x.t \mid v^\varepsilon \cdot s \mid l \rangle^- \rightsquigarrow \langle t[v/x] \mid s \mid l \rangle^\varepsilon$$

$$\langle (\pi_i v)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle v \mid \pi_i^\varepsilon \cdot s \mid l \rangle^- \qquad \langle \langle t_1, t_2 \rangle \mid \pi_i^\varepsilon \cdot s \mid l \rangle^- \rightsquigarrow \langle t_i \mid s \mid l \rangle^\varepsilon$$

$$\langle \delta((v,w),(x,y).t)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle t[v/x, w/y] \mid s \mid l \rangle^\varepsilon \qquad \langle \textbf{new} \mid () \cdot s \mid r_n :: l \rangle^- \rightsquigarrow \langle \iota_1 r_n \mid s \mid l \rangle^+$$

$$\langle \delta((),().t)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle t \mid s \mid l \rangle^\varepsilon \qquad \langle \textbf{new} \mid () \cdot s \mid [] \rangle^- \rightsquigarrow \langle \iota_2() \mid s \mid [] \rangle^+$$

$$\langle \delta(\iota_i v, x_1.t_1, x_2.t_2)^\varepsilon \mid s \mid l \rangle^\varepsilon \rightsquigarrow \langle t_i[v/x_i] \mid s \mid l \rangle^\varepsilon \qquad \langle \textbf{delete} \mid r_n \cdot s \mid l \rangle^- \rightsquigarrow \langle () \mid s \mid r_n :: l \rangle^+$$

Operational semantics - elementary properties

Reduction rules are *deterministic*: polarity annotations block critical pair from ill-typed expressions.

We type stacks following sequent calculus:

$$\frac{}{\star : A \vdash_p A} \qquad \frac{s : B_\varepsilon \vdash_p C \qquad \vdash_p v : A}{v^\varepsilon \cdot s : B \multimap A \vdash_p C} \qquad \frac{s : B_\varepsilon \vdash_p C \qquad x : A \vdash_p t : B}{(x^+.t)^\varepsilon \cdot s : A \vdash_p C}$$

$$\frac{s : A_\varepsilon \vdash_p C}{\pi_1^\varepsilon \cdot s : A \,\&\, B \vdash_p C} \qquad \frac{s : B_\varepsilon \vdash_p C}{\pi_2^\varepsilon \cdot s : A \,\&\, B \vdash_p C}$$

Well-typed commands $\langle \vdash t : A \mid s : A \vdash B \mid l \rangle : B$ enjoy *subject reduction* and *progress*.

Resource safety - definitions

We now want to prove that $\forall \vdash t : W, \forall l, \forall l', \forall v, \langle t \mid \star \mid l \rangle \rightsquigarrow^* \langle v \mid \star \mid l \rangle \Rightarrow l = l'$.

Approach: generalizes typing rules of terms to track their list of resources LR, then show that LR is invariant by reduction.

We need to reason on $LR(t[v/x])$ from $LR(t)$ and $LR(v)$. It can either be $LR(t) +\!\!+ LR(v)$ or $LR(v) +\!\!+ LR(t)$, so we need two substitution lemmas for the leftmost and rightmost variables.

Typing judgments get the shape $\Theta \vdash_o t$ with $\Theta = V; L; V'$ two lists of variables V, V' split by a list of resources L. Then, $LR(t) \stackrel{\text{def}}{=} L$ (omitting types for brevity).

Resource safety - generalized typing

Given $\Theta = V_1; L; V_2$ and $\Theta' = V_3; L'; V_4$, the concatenation $\Theta \mathbin{@} \Theta'$ asserts that either:

- $L = \emptyset$, then $\Theta \mathbin{@} \Theta' \stackrel{\text{def}}{=} V_1 +\!\!+ V_2 +\!\!+ V_3; L'; V_4$.
- $L' = \emptyset$, then $\Theta \mathbin{@} \Theta' \stackrel{\text{def}}{=} V_1; L; V_2 +\!\!+ V_3 +\!\!+ V_4$.
- $V_2 = V_3 = \emptyset$, then $\Theta \mathbin{@} \Theta' \stackrel{\text{def}}{=} V_1; L +\!\!+ L'; V_4$.

$$\frac{}{;[r_n];\vdash_o r_n} \qquad \frac{x, \Theta \vdash_o t}{\Theta \vdash_o \lambda x.t} \qquad \frac{\Theta, x \vdash_o u \qquad \Theta' \vdash_o t}{\Theta \mathbin{@} \Theta' \vdash_o \mathrm{let}\, x = t \,\mathrm{in}\, u} \qquad \dots$$

Stacks are typed similarly, then commands get their list of resources:

$$\frac{;L;\vdash_o s \qquad ;L';\vdash_o t}{L +\!\!+ L' +\!\!+ l' \vdash_o \langle t \mid s \mid l \rangle}$$

Resource safety - proof

We obtain substitution lemmas by induction on t:

- $x, V; L_t; V' \vdash_o t$ and $; L; \vdash_o v$ implies $V; L +\!\!+ L_t; V' \vdash_o t[v/x]$.
- $V; L_t; V', x \vdash_o t$ and $; L; \vdash_o v$ implies $V; L_t +\!\!+ L; V' \vdash_o t[v/x]$.

By case on reduction steps, we have $\forall(l \vdash_o c), \forall(l' \vdash_o c'), c \rightsquigarrow c' \Rightarrow l = l'$.

Expressions $\vdash_o t : W$ in $\boldsymbol{O}$ do not contain resources, hence $l \vdash_o \langle t \mid \star \mid l \rangle$ and $l' \vdash_o \langle v \mid \star \mid l' \rangle$. By transitivity, $\langle t \mid \star \mid l \rangle \rightsquigarrow^* \langle v \mid \star \mid l \rangle$.

For $\boldsymbol{\mathcal{L}}$, we instead use multisets of resources: we obtain that $\exists \sigma. \langle t \mid \star \mid l \rangle \rightsquigarrow^* \langle v \mid \star \mid \sigma(l) \rangle$.

Concrete model - definition

The presheaf model $[R] \to \mathit{Set}$ interpret $\boldsymbol{O}$ types by sets equipped with a list of resources preserved by all operations ($[R]$ the set of lists of resources considered as a discrete category).

The Day construction over $+\!\!+$ yields $\otimes$, $\multimap$ and $\circ\!\!-$. The presheaf structure yields $\&$ and $\oplus$. Resources R are singleton indicators. The terminal object is $[R] \stackrel{\text{def}}{=}$ the initial algebra of $1 \oplus (R \otimes -)$. Resources in command derivations are interpreted by variables.

Finally, the "strong" adjunction $\otimes \dashv \multimap$ yields an ordered call-by-push-value, adapting *A theory of effects and resources: adjunction models and polarised calculi* to the ordered case for adjoint endofunctors that form a strong monad.

Concrete model - properties

The interpretation is sound: $L \vdash c \rightsquigarrow L \vdash c' \Rightarrow [\![c]\!] = [\![c']\!]$.

Three properties coincide for any type A:

- A is central, i.e. in a Drinfeld center.
- A cannot contain resources, i.e. $A(I \neq [\,]) = \emptyset$.
- A is affine, i.e. has a morphism $A \longrightarrow 1$.

$\mathcal{L}$ can be interpreted by replacing lists of resources by multisets (could add isomorphisms instead in the base category).

Error handling

Real-life scenarios involve dealing with errors, especially at resource acquisition (e.g. an allocation failure). E.g. in C, rare case of idiomatic *goto*:

```
    let f = open_file("record.txt");
    if !f return;
    append(f, "starting tasks");
    let g = open_file("output.txt");
    if !g goto err;
    run_tasks(f, g);
    close_file(g);
  err:
    close_file(f);
```

Error-prone: no file ownership, e.g. does `run_tasks` close its file?

Issue of linearity and control

Control effects act on continuations, hence local variables.

E.g. propagating an exception needs to erase them (stack unwinding).

Abstractly: lack of strength of $\mathcal{E} \stackrel{\text{def}}{=} - \oplus E$, cannot bind failible expressions in a linear context.

$\Gamma \otimes \mathcal{E}A \to \mathcal{E}(\Gamma \otimes A) \simeq (\Gamma \otimes A \to (\Gamma \otimes A) \oplus E) \times \Gamma \otimes E \to (\Gamma \otimes A) \oplus E$:

Given by $(\iota_1, \iota_2 \circ \text{"erase } \Gamma\text{"})$: needs a map $\Gamma \to 1$.

Higher-order combinators

Stack-like resource management derived from LISP's `unwind-protect` (Python's `with..as`, C# `using`, ...):

```
with_file("record.txt", λ f.
  append(f, "starting tasks");
  with_file("output.txt", λ g.
    run_tasks(f, g)
  );
);
```

Second-class resources: cannot combine them (e.g. list of files) or return them.

Destructors

C++ *destructors*: effectful erasure maps associated to datatypes, called at the end of their scope.

```
File f{"record.txt"};
append(f, "starting tasks");
File g{"output.txt"};
run_tasks(f, g);
// g then f are then closed by their destructors
```

Derive destructors for collections of resources, closures capturing resources.

Moving resources

C++11: can *move* resources to transfer ownership. Alters their lifetime.

```
File f{"log.txt"};
if init_tasks() == ok {
    append(f, "starting tasks");
    run_tasks(std::move(f));
} else {
    append(f, "failure to init tasks");
}
```

Resource modality - bang

In linear logic: "!" Eilenberg-Moore resolution $\mathcal{C}^! \underset{F^!}{\overset{U^!}{\rightleftarrows}} \mathcal{C}$ yield a resource modality, i.e. U is strong monoidal.

$\mathcal{C}^!$ monoidal structure is cartesian: we have $f : \mathcal{C}^!(\Gamma, \top)$, hence maps $U^! f : \mathcal{C}(U\Gamma, 1)$.

Allows faillible expressions only in contexts from $\mathcal{C}^!$.

Resource modality - destructors

Inspiration from C++: build a resource modality for types Γ with destructors, i.e. effectful erasure maps $\delta : \Gamma \to T1$.

From notes *A resource modality for RAII*: Eilenberg-Moore resolution for the comonad $- \,\&\, T1$ yields the resource modality $C/T1 \underset{(A \,\&\, T1, \pi_2) \leftarrow\!\shortmid A}{\overset{(A,\delta) \mapsto A}{\rightleftarrows}} C$.

Then, define $T\mathcal{E}$ relaxed strength maps $U(\Gamma, \delta) \otimes T\mathcal{E}A \to T\mathcal{E}(U(\Gamma, \delta) \otimes A)$.

It requires maps $\Gamma \otimes E \to TE$, obtained from δ and $\mathrm{swap} : \Gamma \otimes E \to E \otimes \Gamma$.

Call-by-push-value with destructors

Adapt construction to the ordered case: require E to be central.

Define an ordered call-by-push-value calculus $\boldsymbol{O}_{\mathcal{E}}$ between destructible values and failible computations. Add a **move** operation to obtain the full calculus $\boldsymbol{O}_{\mathcal{E},\mathbf{move}}$.

Semantics given by translation in $\mathcal{L}$, leaving T implicit:

$$\begin{array}{ccc} \boldsymbol{O}_{\mathcal{E}} & \xrightarrow{[\![-]\!]} & \boldsymbol{O} \\ \cap & & \cap \\ \boldsymbol{O}_{\mathcal{E},\mathbf{move}} & \xrightarrow{[\![-]\!]} & \mathcal{L} \end{array}$$

Call-by-push-value with destructors - grammar

Extends $\mathcal{L}$: reuse grammar without annotations, then adds

- "**drop**" to destruct a value.
- "**move**(x, y) in t" to exchange variables.
- "**raise**" to throw an exception.
- "$\mathrm{try}\, x \Leftarrow t \,\mathrm{in}\, u \,\mathrm{unless}\, e \Rightarrow u'$" to catch exceptions.

Replace R by R_d in types.

Call-by-push-value with destructors - typing rules

Fix a central type E, remove **new** and **delete** typing rules from $\mathcal{O}$, then add the following rules:

$$\frac{}{\vdash \mathbf{drop} : 1 \circ\!\!- A}\ \textbf{drop}$$

$$\frac{}{\vdash \mathbf{new} : R_d \circ\!\!- 1}\ \textbf{new}$$

$$\frac{}{\vdash \mathbf{raise} : A \circ\!\!- E}\ \textbf{raise}$$

$$\frac{\Gamma, x : A, y : B, \Gamma' \vdash t : C}{\Gamma, y : B, x : A, \Gamma' \vdash \mathbf{move}(x, y) \text{ in } t : C}\ \textbf{move}$$

$$\frac{\Delta \vdash t : P \qquad \Gamma, x : P \vdash u : A \qquad \Gamma, e : E \vdash u' : A}{\Gamma, \Delta \vdash \mathrm{try}\ x \Leftarrow t \text{ in } u \text{ unless } e \Rightarrow u' : A}\ \text{try}$$

Call-by-push-value with destructors - translation (1)

Fix a definable value $\vdash \mathsf{new\_fail} : E$ for allocation failures.

Follows call-by-push-value semantics with shifts $\Uparrow A \stackrel{\text{def}}{=} A \oplus E$ and $\Downarrow A \stackrel{\text{def}}{=} A \,\&\, 1$.

$$1^+ \stackrel{\text{def}}{=} 1 \qquad \mathbf{drop}_1 \stackrel{\text{def}}{=} \lambda v.v$$

$$R_d^+ \stackrel{\text{def}}{=} R \qquad \mathbf{drop}_{R_d} \stackrel{\text{def}}{=} \lambda r.\,\mathbf{delete}\,r$$

$$(A \otimes B)^+ \stackrel{\text{def}}{=} A^+ \otimes B^+ \qquad \mathbf{drop}_{(A\otimes B)} \stackrel{\text{def}}{=} \lambda p.\delta(p,(a,b).\,\mathbf{drop}_B\,b;\mathbf{drop}_A\,a)$$

$$(A \oplus B)^+ \stackrel{\text{def}}{=} A^+ \oplus B^+ \qquad \mathbf{drop}_{(A\oplus B)} \stackrel{\text{def}}{=} \lambda s.\delta(s,a.\,\mathbf{drop}_A\,a,b.\,\mathbf{drop}_B\,b)$$

$$N^+ \stackrel{\text{def}}{=} \Downarrow A^- \qquad \mathbf{drop}_N \stackrel{\text{def}}{=} \lambda a.\pi_2 a$$

$$(B \multimap A)^- \stackrel{\text{def}}{=} B^- \multimap A^+$$

$$(A \,\&\, B)^- \stackrel{\text{def}}{=} A^- \,\&\, B^- \qquad \star^+ \stackrel{\text{def}}{=} \star$$

$$P^- \stackrel{\text{def}}{=} \Uparrow A^+ \qquad (\Gamma, x : A)^+ \stackrel{\text{def}}{=} \Gamma^+, x : A^+$$

Call-by-push-value with destructors - translation (2)

Translate derivations of values and expressions by mutual induction:

- $[\![-]\!]_{val} : (\Gamma \vdash^{\mathcal{O}_{\mathcal{E},\mathbf{move}}}_{val} A) \to (\Gamma^+ \vdash^{\mathcal{L}}_{val} A^+)$.
- $[\![-]\!]_{expr} : (\Gamma \vdash^{\mathcal{O}_{\mathcal{E},\mathbf{move}}}_{expr} A) \to (\Gamma^+ \vdash^{\mathcal{L}}_{val} \Downarrow A^-)$.

Permuting variables is only done for **move**, hence preserving the ordered fragment.

Get strength from the expression $\Gamma^+, e : E \vdash \mathrm{unwind}_\Gamma(e) : E$ defined by induction on Γ:

$$\mathrm{unwind}_\star(e) \stackrel{\mathrm{def}}{=} e$$

$$\mathrm{unwind}_{\Gamma,x:A}(e) \stackrel{\mathrm{def}}{=} \mathrm{let}\ p = \mathrm{swap}^A_E(x, e)\ \mathrm{in}\ \delta(p, (e, x).\ \mathbf{drop}_A\, x; \mathrm{unwind}_\Gamma(e))$$

Call-by-push-value with destructors - properties

Positive expressions $t : P$ receive an eagerly discarded destructor ($[\![t]\!] : (P \oplus E)\,\&\,1$). Hence, given $\vdash^{\boldsymbol{O}_{\mathcal{E},\mathbf{move}}} t : P$, run $\langle \pi_1 [\![t]\!] : P \oplus E \mid \star \mid l \rangle$.

Resource-safety properties on $\boldsymbol{O}$ (resp $\boldsymbol{\mathcal{L}}$) apply to $\boldsymbol{O}_{\mathcal{E}}$ (resp $\boldsymbol{O}_{\mathcal{E},\mathbf{move}}$):

- $\forall \vdash^{\boldsymbol{O}_{\mathcal{E}}} t : W, \forall l, \forall l', \forall v, \langle \pi_1 [\![t]\!] : W \oplus E \mid \star \mid l \rangle \rightsquigarrow^* \langle v \mid \star \mid l \rangle \Rightarrow l = l'$.
- $\forall \vdash^{\boldsymbol{O}_{\mathcal{E},\mathbf{move}}} t : W, \forall l, \forall l', \forall v, \langle \pi_1 [\![t]\!] : W \oplus E \mid \star \mid l \rangle \rightsquigarrow^* \langle v \mid \star \mid l \rangle \Rightarrow \exists \sigma, \sigma(l) = l'$.

As $W \oplus E$ is also central.

Conclusion

Our results suggest to model destructors with $\boldsymbol{O}_{\mathcal{E}}$, and destructors + **move** with $\boldsymbol{O}_{\mathcal{E},\textbf{move}}$.

$\boldsymbol{O}_{\mathcal{E},\textbf{move}}$ has aspects of affine, linear and ordered logic:

affine	in terms of expressiveness and available control effects
linear	in terms manipulation of values and resource-safety
ordered	in terms of proof-relevance and type isomorphisms