3 CORE CALCULUS

We present our core calculus in three steps. First, we present the basic calculus only for linearity. This describes the essence of our calculus and most rules carry over unchanged. Then we add uniqueness and locality, which only requires us to change the VAR and LAM rules.

To see why these two rules are special, consider the following discussion: Traditionally, we would expect the VAR rule to only check that the variable is in the context with the correct type. This then imposes an extra constraint on the ABS rule, to ensure that only non-linear variables can enter a non-linear closure:

$$\frac{}{\Gamma_1, x : \tau \mathbin{@} \mu, \Gamma_2 \vdash x : \tau \mathbin{@} \mu}\ \textsc{var} \qquad \frac{\Gamma, x : \tau_1 \mathbin{@} \mu_1 \vdash e : \tau_2 \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} \textsc{once}}\ \textsc{abs-once}$$

$$\frac{\mathrm{dup}(\Gamma), x : \tau_1 \mathbin{@} \mu_1 \vdash e : \tau_2 \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} \textsc{many}}\ \textsc{abs-many}$$

where dup removes all linear variables from Γ. However, in our presentation, we will not delete variables from the context directly, but instead defer this decision to the VAR rule. Concretely, in the ABS rule, we put a lock $🔒_\mu$ into the context with μ the mode of the closure. Then we only introduce a variable from the context if it can pass through all of the locks that were introduced later.

$$\frac{\mu \leq \mu_i \text{ for all } 🔒_{\mu_i} \in \Gamma_2}{\Gamma_1, x : \tau \mathbin{@} \mu, \Gamma_2 \vdash x : \tau \mathbin{@} \mu}\ \textsc{var} \qquad \frac{\Gamma, 🔒_\mu, x : \tau_1 \mathbin{@} \mu_1 \vdash e : \tau_2 \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} \mu}\ \textsc{abs}$$

Most importantly, this means that the μ on the lock can be a inferred later, where the conditions in the VAR rule become constraints that can be passed to a subtyping solver. Thus, programmers do not have to specify the mode μ of a closure in the ABS rule, but can instead let the type-checker infer it.

3.1 Linearity

In this section, we only consider the calculus for linearity and thus instantiate μ thus to the mode many $<$ once. In our contexts, we include variables $x : \tau @ \mu$, annotated with their mode μ, unused variables $x : -$ (see below) and locks $🔒_\mu$ of mode μ. In the types, we include modes on both the argument and the return of the function type. Otherwise, our syntax is standard.

$$\begin{aligned}
b &::= - \mid \tau \mathbin{@} \mu \\
\Gamma &::= \cdot \mid \Gamma, x : b \mid \Gamma, 🔒_\mu \\
\tau &::= \tau \mathbin{@} \mu \to \tau \mathbin{@} \mu \mid \tau \times \tau \mid \mathrm{Unit} \mid \tau + \tau \\
e &::= x \mid \lambda x.\, e \mid e\ e \mid (e, e) \mid \mathrm{fst}\ e \mid \mathrm{snd}\ e \\
&\quad \mid () \mid e;\, e \mid \mathrm{inl}\ e \mid \mathrm{inr}\ e \\
&\quad \mid \mathrm{case}\ e\ \{\, \mathrm{inl}\ x \to e; \mathrm{inr}\ x \to e \,\}
\end{aligned}$$

The ordering on modes extends to an ordering of bindings and contexts:

$$\begin{aligned}
&&&- \leq \tau \mathbin{@} \mu \\
\mu_2 \leq \mu_1 \quad &\Longrightarrow \quad &&\tau \mathbin{@} \mu_1 \leq \tau \mathbin{@} \mu_2 \\
\\
&&&\cdot \leq \cdot \\
\Gamma_1 \leq \Gamma_2, b_1 \leq b_2 \quad &\Longrightarrow \quad &&(\Gamma_1, x : b_1) \leq (\Gamma_2, x : b_2) \\
\Gamma_1 \leq \Gamma_2 \quad &\Longrightarrow \quad &&(\Gamma_1, 🔒_\mu) \leq (\Gamma_2, 🔒_\mu)
\end{aligned}$$

This ordering on contexts allows us to weaken the context with stronger modes:

$$\frac{\Gamma_1 \leq \Gamma_2 \qquad \Gamma_1 \vdash e : \tau \mathbin{@} \mu_1 \qquad \mu_1 \leq \mu_2}{\Gamma_2 \vdash e : \tau \mathbin{@} \mu_2}\ \textsc{sub}$$

On contexts, we define an addition operation $\Gamma_1 \fatsemi \Gamma_2$, which joins the contexts. Since we have locks in our contexts, we need to be careful to preserve the order of the locks. To do this, we assume that the two contexts to be joined contain the same locks and variables in the same order. Our join operation is then defined inductively as:

$$\begin{aligned}
\cdot \fatsemi \cdot &= \cdot \\
(\Gamma_1, x : -) \fatsemi (\Gamma_2, x : -) &= (\Gamma \fatsemi \Gamma_2), x : - \\
(\Gamma_1, x : \tau \mathbin{@} \mu) \fatsemi (\Gamma_2, x : -) &= (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@} \mu \\
(\Gamma_1, x : -) \fatsemi (\Gamma_2, x : \tau \mathbin{@} \mu) &= (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@} \mu \\
(\Gamma_1, x : \tau \mathbin{@} \bot) \fatsemi (\Gamma_2, x : \tau \mathbin{@} \bot) &= (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@} \mathsf{many} \\
(\Gamma_1, \text{🔒}_\mu) \fatsemi (\Gamma_2, \text{🔒}_\mu) &= (\Gamma \fatsemi \Gamma_2), \text{🔒}_\mu
\end{aligned}$$

Since we allow a subsumption rule, we write $\top$ or $\bot$ if any mode is acceptable. In particular, we write an unspecified mode as $\top$ if it occurs in the mode of a term in a premise, or in the context of the conclusion. Dually, we write an unspecified mode as $\bot$ if it occurs in the mode of the term in the conclusion, or in the context of a premise. The other rules for our syntax are as follows:

$$\frac{\Gamma_1 \vdash e_1 : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} \mu_3 \qquad \Gamma_2 \vdash e_2 : \tau_1 \mathbin{@} \mu_1}{\Gamma_1 \fatsemi \Gamma_2 \vdash e_1\, e_2 : \tau_2 \mathbin{@} \mu_2}\ \textsc{app} \qquad \frac{\Gamma_1 \vdash e_1 : \tau_1 \mathbin{@} \mu \qquad \Gamma_2 \vdash e_2 : \tau_2 \mathbin{@} \mu}{\Gamma_1 \fatsemi \Gamma_2 \vdash (e_1, e_2) : \tau_1 \times \tau_2 \mathbin{@} \mu}\ \textsc{pair}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2 \mathbin{@} \mu}{\Gamma \vdash \mathrm{fst}\ e : \tau_1 \mathbin{@} \mu}\ \textsc{fst} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2 \mathbin{@} \mu}{\Gamma \vdash \mathrm{snd}\ e : \tau_2 \mathbin{@} \mu}\ \textsc{snd} \qquad \frac{}{\Gamma \vdash () : \mathrm{Unit} \mathbin{@} \mu}\ \textsc{unit}$$

$$\frac{\Gamma_1 \vdash e_1 : \mathrm{Unit} \mathbin{@} \mu_1 \qquad \Gamma_2 \vdash e_2 : \tau \mathbin{@} \mu_2}{\Gamma_1 \fatsemi \Gamma_2 \vdash e_1; e_2 : \tau \mathbin{@} \mu_2}\ \textsc{seq} \qquad \frac{\Gamma \vdash e : \tau_1 \mathbin{@} \mu}{\Gamma \vdash \mathrm{inl}\, e : \tau_1 + \tau_2 \mathbin{@} \mu}\ \textsc{inl} \qquad \frac{\Gamma \vdash e : \tau_1 \mathbin{@} \mu}{\Gamma \vdash \mathrm{inr}\, e : \tau_1 + \tau_2 \mathbin{@} \mu}\ \textsc{inr}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 + \tau_2 \mathbin{@} \mu_1 \qquad \Gamma_2, x_1 : \tau_1 \mathbin{@} \mu_1 \vdash e_2 : \tau_3 \mathbin{@} \mu_2 \qquad \Gamma_2, x_2 : \tau_2 \mathbin{@} \mu_1 \vdash e_3 : \tau_3 \mathbin{@} \mu_2}{\Gamma_1 \fatsemi \Gamma_2 \vdash \mathrm{case}\ e_1\, \{\, \mathrm{inl}\, x_1 \to e_2; \mathrm{inr}\, x_2 \to e_3 \,\} : \tau_3 \mathbin{@} \mu_2}\ \textsc{case}$$

3.2 Uniqueness

In this section, we add a uniqueness mode. For this, we instantiate μ to a tuple (o, u) of a linearity mode $o = \mathsf{many} < \mathsf{once}$ and a uniqueness mode $u = \mathsf{unique} < \mathsf{shared}$. To make the uniqueness meaningful, we add a type of space credits ♣ (we avoid the $\diamond$ symbol to avoid confusion with the diamond modality). We also add a new expression to extract the values of a pair alongside its space credit and the expression reuse x with (e_1, e_2), which allows us to reuse the space credit of x to allocate a new

$$\begin{aligned}
\tau &::= \cdots \mid \clubsuit \\
e &::= \cdots \mid \mathrm{let}\,(x, x, x) = e\,\mathrm{in}\,e \\
&\quad \mid \mathrm{reuse}\ x\ \mathrm{with}\ (e, e)
\end{aligned}$$

pair. Note that we keep the FST and SND rules and thus it is not required to destruct a pair with the UNPAIR rule.

In the context join operation, we replace the case where a variable occurs on both sides, to additionally request that the variable be shared. This case may give the variable any uniqueness mode u to allow for subsumption.

$$(\Gamma_1, x : \tau \mathbin{@} (\bot, \texttt{shared})) \fatsemi (\Gamma_2, x : \tau \mathbin{@} (\bot, \texttt{shared})) = (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@} (\texttt{many}, \top)$$

Compared to the last section, we only need to change the VAR and ABS rules. Perhaps surprisingly, we do not need to record the uniqueness of a closure on a lock. This is, since a closure capturing a unique variable can only be used once, so it has to be `once` (rather than `unique`). Conversely, we have to check in the VAR rule that a variable can only be used uniquely, if this does not happen under a `many` lock. We define $\texttt{many}^\dagger = \texttt{shared}$ and $\texttt{once}^\dagger = \texttt{unique}$.

$$\frac{o \leq o_i \text{ and } o_i^\dagger \leq u \text{ for all } \blacksquare_{o_i} \in \Gamma_2}{\Gamma_1, x : \tau \mathbin{@} (o, u), \Gamma_2 \vdash x : \tau \mathbin{@} (o, u)}\ \textsc{var} \qquad \frac{\Gamma, \blacksquare_o, x : \tau_1 \mathbin{@} \mu_1 \vdash e : \tau_2 \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} (o, \bot)}\ \textsc{abs}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \times \tau_2 \mathbin{@} \mu_1 \qquad \Gamma_2, c : \clubsuit \mathbin{@} \mu_1, x_1 : \tau_1 \mathbin{@} \mu_1, x_2 : \tau_2 \mathbin{@} \mu_1 \vdash e_2 : \tau_3 \mathbin{@} \mu_2}{\Gamma_1 \fatsemi \Gamma_2 \vdash \text{let } (c, x_1, x_2) = e_1 \text{ in } e_2 : \tau_3 \mathbin{@} \mu_2}\ \textsc{unpair}$$

$$\frac{\Gamma_1 \vdash e_1 : \clubsuit \mathbin{@} (\top, \texttt{unique}) \qquad \Gamma_2 \vdash e_2 : \tau_1 \mathbin{@} \mu \qquad \Gamma_3 \vdash e_3 : \tau_2 \mathbin{@} \mu}{\Gamma_1 \fatsemi \Gamma_2 \fatsemi \Gamma_3 \vdash \text{reuse } e_1 \text{ with } (e_2, e_3) : \tau_1 \times \tau_2 \mathbin{@} \mu}\ \textsc{update}$$

3.3 Locality and Borrowing

In this section, we extend the calculus by stack allocation and borrowing. We instantiate μ to a triple (o, u, l) of a linear mode o, uniqueness mode u and locality mode $l = \texttt{global} < \texttt{local}$. In contexts, we now also track borrowed variables $x : \tau \mathbin{@}^{\mathbb{B}} o$. A variable is borrowed if it was used as local and shared inside a region (which it can not escape). Thus, the memory underlying a borrowed variable was not changed and has not acquired any further references. We extend the ordering as follows:

$$b ::= \cdots \mid \tau \mathbin{@}^{\mathbb{B}} o$$
$$e ::= \cdots \mid \text{region } e$$

$$\begin{aligned} - &\leq \tau \mathbin{@}^{\mathbb{B}} \bot \\ \tau \mathbin{@}^{\mathbb{B}} o &\leq \tau \mathbin{@} (o, \bot, \bot) \end{aligned}$$

We define a new region-closing operator $|\Gamma|$ on contexts, which marks all variables which were used as shared and local as borrowed.

$$
\begin{aligned}
|\cdot| &= \cdot \\
|\Gamma, x : \tau \mathbin{@} (o, \texttt{shared}, \texttt{local})| &= |\Gamma|, x : \tau \mathbin{@}^{\mathbb{B}} o \\
|\Gamma, x : b| &= |\Gamma|, x : b \\
|\Gamma, \blacksquare_\mu| &= |\Gamma|, \blacksquare_\mu
\end{aligned}
$$

This allows us to recover their uniqueness at the end of the scope. We again extend the context join operator. The first case is just as before (but now including uniqueness), but the latter cases differ: If a variable is used as borrowed after it is used as owned, we gain no benefit from the borrowing since the first use has the ability to change the underlying memory. However, if a variable is used as borrowed before it is used as owned, we can safely borrow the memory and continue to use it as owned.

$$
\begin{aligned}
(\Gamma_1, x : \tau \mathbin{@} (\bot, \texttt{shared}, l)) \fatsemi (\Gamma_2, x : \tau \mathbin{@} (\bot, \texttt{shared}, l)) &= (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@} (\texttt{many}, \top, l) \\
(\Gamma_1, x : \tau \mathbin{@}^{\mathbb{B}} o) \fatsemi (\Gamma_2, x : \tau \mathbin{@} (o, u, l)) &= (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@} (o, u, l) \\
(\Gamma_1, x : \tau \mathbin{@}^{\mathbb{B}} \bot) \fatsemi (\Gamma_2, x : \tau \mathbin{@}^{\mathbb{B}} \bot) &= (\Gamma \fatsemi \Gamma_2), x : \tau \mathbin{@}^{\mathbb{B}} \texttt{many}
\end{aligned}
$$

Again, we only have to change the VAR and ABS rules. Locality works similarly to linearity: both are comonads and we add both to the locks. We allow the ABS rule to introduce the variable as either borrowed or owned, but demand that only owned variables can be used in the VAR rule.

$$
\frac{o \leq o_i \text{ and } o_i^\dagger \leq u \text{ and } l \leq l_i \text{ for all } \blacksquare_{(o_i, l_i)} \in \Gamma_2}{\Gamma_1, x : \tau \mathbin{@} (o, u, l), \Gamma_2 \vdash x : \tau \mathbin{@} (o, u, l)} \textsc{var}
$$

$$
\frac{\Gamma, \blacksquare_{(o,l)}, x : \tau_1 \mathbin{@} \mu_1 \vdash e : \tau_2 \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} (o, \bot, l)} \textsc{abs} \qquad \frac{\Gamma \vdash e : \tau \mathbin{@} (o, u, \texttt{global})}{|\Gamma| \vdash \text{region } e : \tau \mathbin{@} (o, u, \texttt{global})} \textsc{region}
$$

Care needs to be taken when combining stack allocation with in-place update. As stated above, it would be possible to extract a heap allocated space credit and use it to create a new local pair that stores values on the stack. While this makes sense semantically, it will create a pointer from the heap to the stack, which is problematic for a tracing garbage collector. Therefore, we disallow this combination in practice.

3.4 Boxing and unboxing

Unlike in systems such as type qualifiers, our pairs require that both components have the same mode as the pair itself. This seems sensible: A pair of once components can not be many itself. However, this makes it impossible to use a pair containing both a once and a many component, since the system above has no way to extract a many value out from a once value. To fix this, we introduce a new type $\Box_\mu$ which captures a value at mode μ. Irrespective of the mode of the box, we can access the value inside the box as μ.

$$
\begin{aligned}
\tau &::= \cdots \mid \Box_\mu \tau \\
e &::= \cdots \mid \text{box } e \mid \text{unbox } e
\end{aligned}
$$

$$\frac{\Gamma \vdash e : \tau \mathbin{@} (o,u,l) \qquad o^\dagger \leq u}{\Gamma \vdash \text{box}\, e : \square_{(o,u,l)} \tau \mathbin{@} (o,\bot,l)}\ \textsc{box} \qquad \frac{\Gamma \vdash e : \square_{\mu_1} \tau \mathbin{@} \mu_2}{\Gamma \vdash \text{unbox}\, e : \tau \mathbin{@} \mu_1}\ \textsc{unbox}$$

Note that these rules follow directly from the ABS rule, since we can simulate box e by $(\lambda x.\lambda y.x)e$ and unbox e by applying a unit value e ().

3.5 More precise checking of usage

In this section, we want to consider an extension to the calculus to check examples like the below:

```
let f x =
   let (x1, x2) = x in
   if ... then unique_use(x1)
          else shared_use(x); shared_use(x)
   ...
   shared_use(x2)
```

Here, we use `x2` twice, but use `x1` only once. Consequently, given a unique `x`, our analysis should accept this program. However, the analysis outlined above is too coarse and rejects this program, since it counts the unpairing of `x` as a use. While we could fix this program by replacing the shared uses of `x` by uses of a new pair `(x1, x2)`, this would require an allocation since any space credit from the original `x` can only be used for one of the newly constructed pairs.

To fix this, we will use paths $y : b$ rather than variables in our typing context. A path extends a variable with a series of selectors with a syntax given by: $y ::= x \mid y.\& \mid y.inl \mid y.inr \mid y.fst \mid y.snd$. Here, $y.\&$ denotes the memory cell underlying y, while inl, inr, fst, snd denote the left/right component of a sum or the first/second component of a pair respectively. We denote the type of $y.\&$ by $\text{mem}(\tau)$ if y is of type τ.

For contexts, we demand now that the modes of paths obey an ordering: We define $parent(x) = x$ and $parent(y.sel) = y$ for any selector $sel \in \{\&, inl, inr, fst, snd\}$. Then we define the syntax of well-formed contexts as:

$$\frac{}{\cdot \text{ well-formed}}\ \textsc{empty} \qquad \frac{\Gamma \text{well-formed}}{\Gamma, \blacksquare_\mu \text{ well-formed}}\ \textsc{lock}$$

$$\frac{\Gamma \text{ well-formed} \qquad parent(y) = y \text{ or } (parent(y) : b_2 \in \Gamma \text{ and } b_2 \leq b)}{\Gamma, y : b \text{ well-formed}}\ \textsc{var}$$

While paths can not be written in the source program, we can use them in the UNPAIR-PATH and CASE-PATH rules to record if a match happened on a path (instead of a general term). In that case, we substitute the path for the original variable name. In the VAR rule, we now also consider paths. In the ABS rule, we introduce all possible paths at the same time where

$$\begin{aligned}
\Gamma_{y,\tau_1+\tau_2,\mu} &= y : (\tau_1 + \tau_2) \mathbin{@} \mu, y.\& : \text{mem}(\tau_1 + \tau_2) \mathbin{@} \mu, \Gamma_{y.inl,\tau_1,\mu}, \Gamma_{y.inr,\tau_2,\mu} \\
\Gamma_{y,\tau_1\times\tau_2,\mu} &= y : (\tau_1 \times \tau_2) \mathbin{@} \mu, y.\& : \text{mem}(\tau_1 \times \tau_2) \mathbin{@} \mu, \Gamma_{y.fst,\tau_1,\mu}, \Gamma_{y.snd,\tau_2,\mu} \\
\Gamma_{y,\tau,\mu} &= y : \tau \mathbin{@} \mu \text{ else}
\end{aligned}$$

Then, we do not have to introduce paths when matching on a path in the UNPAIR-PATH and CASE-PATH rules. Instead, the sub-paths are already in the context and we can use them directly.

$$\frac{o \leq o_i \text{ and } o_i^\dagger \leq u \text{ and } l \leq l_i \text{ for all } \blacksquare_{(o_i,l_i)} \in \Gamma_2}{\Gamma_1, y :^{\mathbb{O}} \tau \mathbin{@} (o,u,l), \Gamma_2 \vdash y : \tau \mathbin{@} (o,u,l)} \textsc{var}$$

$$\frac{\Gamma, \blacksquare_{(o,l)}, \Gamma_{x,\tau_1,\mu_1} \vdash e : \tau_2 \mathbin{@} \mu_2}{\Gamma \vdash \lambda x.\, e : (\tau_1 \mathbin{@} \mu_1 \to \tau_2 \mathbin{@} \mu_2) \mathbin{@} (o, \bot, l)} \textsc{abs}$$

$$\frac{y.\& : \mathrm{mem}(\tau_1 \times \tau_2) \mathbin{@}^{\mathbb{B}} \top \in \Gamma \quad \Gamma \vdash e_2[y.\&/c, y.fst/x_1, y.snd/x_2] : \tau_3 \mathbin{@} \mu}{\Gamma \vdash \mathrm{let}\,(c, x_1, x_2) = y \,\mathrm{in}\, e_2 : \tau_3 \mathbin{@} \mu} \textsc{unpair-path}$$

$$\frac{y.\& : \mathrm{mem}(\tau_1 + \tau_2) \mathbin{@}^{\mathbb{B}} \top \in \Gamma \quad \Gamma \vdash e_2[y.inl/x_1] : \tau_3 \mathbin{@} \mu \quad \Gamma \vdash e_3[y.inr/x_2] : \tau_3 \mathbin{@} \mu}{\Gamma \vdash \mathrm{case}\ y\ \{\,\mathrm{inl}\ x_1 \to e_2; \mathrm{inr}\ x_2 \to e_3\,\} : \tau_3 \mathbin{@} \mu} \textsc{case-path}$$

Finally, we want to consider one example where our new extension is more precise than many existing systems. Consider the following program:

```
let f x =
  let y1 = match x with
           | Left(x1) -> unique_use(x1)
           | Right(x2) -> ()
           end
  let y2 = match x with
           | Left(x1) -> ();
           | Right(x2) -> unique_use(x2)
           end
  ...
```

Since we introduce the paths $x1 = x.inl$ and $x2 = x.inr$ at the start of the function, we can see that both are only used once and the program type-checks. In contrast, if we introduced the paths only at the match-statements, we would be forced to conclude that a unique use of $x1$ implies a unique use of x, which disallows the second match-statement.