I wanted to try out making a programming language vaguely corresponding to the double category **Rel**.

You have a calculus of "values" corresponding to one edge of **Rel** and a sort of relational calculus of "predicates" corresponding to the other edge. Satisfies judgements stating a value satisfies a predicate ought to correspond to squares.

The language of "predicates" is based off the Simply Typed Lambda Calculus (in category theory terms is closed/has exponential objects) and the language of "values" has product types (in category theory terms is Cartesian.)

This is the core framework. I've been thinking about further extensions but I want to see how far I can get characterizing **Rel** with as few language features as possible. Later on I give a few possible extensions.

Core Calculus

Types $t ::= \top \mid t \times t$

Sorts $s ::= t? \mid t!$

Values $v ::= x \mid \mathbf{fst}(v) \mid \mathbf{snd}(v) \mid (v, v)$

Predicates $p ::= x \mid p\, p \mid \mu(x\colon t?).p$

Environment $\Gamma ::= \cdot\Gamma, \mid x\colon s$

Substitutions $\sigma ::= \cdot\sigma, \mid v \models x$

I need a better name for the abstraction for predicates. It's a little like the μ abstraction $\bar{\lambda}\mu\tilde{\mu}$ but different.

Typing judgments.

$$\frac{}{\Gamma \vdash \mathbf{tt}\colon \top!}$$
$$\frac{\Gamma \vdash v\colon t_0 \times t_1!}{\Gamma \vdash \mathbf{fst}(v)\colon t_0!}$$
$$\frac{\Gamma \vdash v\colon t_0 \times t_1!}{\Gamma \vdash \mathbf{snd}(v)\colon t_1!}$$
$$\frac{\Gamma \vdash v_0\colon t_0! \quad \Gamma \vdash v_1\colon t_1!}{\Gamma \vdash (v_0, v_1)\colon t_0 \times t_1!}$$
$$\frac{\Gamma \vdash p_0\colon t_0 \times t_1? \quad \Gamma \vdash p_1\colon t_0?}{\Gamma \vdash p_0\, p_1\colon t_1?}$$
$$\frac{\Gamma, x\colon t_0? \vdash p\colon t_1?}{\Gamma \vdash \mu x\colon t_0?.p\colon t_0 \times t_1?}$$

When a value satisfies a predicate (I need a better symbol here.)

$$\frac{v_0 \models p\ [\sigma]}{\mathbf{fst}(v_0, v_1) \models p\ [\sigma]}$$
$$\frac{v_1 \models p\ [\sigma]}{\mathbf{snd}(v_0, v_1) \models p\ [\sigma]}$$
$$\frac{v \models [x\colon\ = p_1]p_0\ [\sigma]}{v \models (\mu(x\colon t?).p_0)\, p_1\ [\sigma]}$$
$$\frac{v_1 \models p\ [\sigma,\ v_0 \models x]}{(v_0, v_1) \models \mu(x\colon t?).p\ [\sigma]}$$

Example

$$(v, v) \models \mu(x\colon t?).x\ [\sigma]$$

Transposition

$$\mathbf{M}(A\colon\ \mathbf{type}).\mu(p\colon A \times A \times \top)(x\colon A)(y\colon A).p\, y\, x$$

Disjoint Unions

I am fairly confident in a simple extension to disjoint unions of sets which are sum types in the value calculus and product types in the predicate calculus.

Types $t ::= \ldots \mid \bot \mid t + t$

Values $v ::= \ldots \mid \mathbf{inl}(t, v) \mid \mathbf{inr}(t, v) \mid$ $\mathbf{match}\ v \begin{cases} v & \leftarrow \mathbf{inl}(x) \\ v & \leftarrow \mathbf{inr}(x) \end{cases}$

Predicates $p ::= \ldots \mid \mathbf{false} \mid \mathbf{left}(p) \mid \mathbf{right}(p) \mid [p; p]$

Typing judgments.

$$\frac{\Gamma \vdash v \colon t_0!}{\Gamma \vdash \mathbf{inl}(t_1, v) \colon t_0 + t_1!}$$
$$\frac{\Gamma \vdash v \colon t_1!}{\Gamma \vdash \mathbf{inr}(t_0, v) \colon t_0 + t_1!}$$
$$\frac{\Gamma \vdash v_0 \colon t_0 + t_1! \; \Gamma,\, x_0 \colon t_0! \vdash v_1 \colon t_2! \; \Gamma,\, x_1 \colon t_1! \vdash v_2 \colon t_2!}{\Gamma \vdash \mathbf{match}\ v_0 \begin{cases} v_1 & \leftarrow \mathbf{inl}(x_0) \\ v_2 & \leftarrow \mathbf{inr}(x_1) \end{cases} \colon t_2!}$$

$$\frac{}{\Gamma \vdash \mathbf{false} \colon \bot?}$$
$$\frac{\Gamma \vdash p \colon t_0 + t_1?}{\Gamma \vdash \mathbf{left}(p) \colon t_0?}$$
$$\frac{\Gamma \vdash p \colon t_0 + t_1?}{\Gamma \vdash \mathbf{right}(p) \colon t_1?}$$
$$\frac{\Gamma \vdash p_0 \colon t_0? \quad \Gamma \vdash p_1 \colon t_1?}{\Gamma \vdash [p_0; p_1] \colon t_0 + t_1?}$$

Satisfies

$$\frac{[x_0 := v_0]v_1 \models p\ [\sigma]}{\mathbf{match}\ \mathbf{inl}(t, v_0) \begin{cases} v_1 & \leftarrow \mathbf{inl}(x_0) \\ v_2 & \leftarrow \mathbf{inr}(x_1) \end{cases} \models p\ [\sigma]}$$
$$\frac{[x_1 := v_0]v_2 \models p\ [\sigma]}{\mathbf{match}\ \mathbf{inr}(t, v_0) \begin{cases} v_1 & \leftarrow \mathbf{inl}(x_0) \\ v_2 & \leftarrow \mathbf{inr}(x_1) \end{cases} \models p\ [\sigma]}$$
$$\frac{v \models p_0\ [\sigma]}{v \models \mathbf{left}([p_0; p_1])\ [\sigma]}$$
$$\frac{v \models p_1\ [\sigma]}{v \models \mathbf{right}([p_0; p_1])\ [\sigma]}$$
$$\frac{v \models p_0\ [\sigma]}{\mathbf{inl}(t, v) \models [p_0; p_1]\ [\sigma]}$$
$$\frac{v \models p_1\ [\sigma]}{\mathbf{inr}(t, v) \models [p_0; p_1]\ [\sigma]}$$

Dependent Sums

If product of sets becomes an exponential in the predicate calculus then dependent sums ought to become like Π types. So the predicate calculus effectively becomes like System-F.

Some things become awkward to interpret here though.

I also really can't figure out unpacking. It's messy if you don't want full dependent types.

Types $t ::= \ldots \mid x \mid \Sigma(x\colon\ \mathbf{type}).t$

Sorts $s ::= \ldots \mid \mathbf{type}$

Values $v ::= \ldots \mid \langle x\colon\ = t, v\rangle$

Predicates $p ::= \ldots \mid p\, t \mid \mathbf{M}(x\colon\ \mathbf{type}).p$

Substitutions $\sigma ::= \ldots \sigma, \mid t \models x$

Not really good at the typing judgements for dependent sum types.

$$\frac{\Gamma \vdash t_0 \colon\ \mathbf{type} \quad \Gamma,\, x\colon\ \mathbf{type} \vdash v \colon t_1!}{\Gamma \vdash \langle x := t_0, v\rangle \colon \Sigma(x\colon\ \mathbf{type}).t_0!}$$
$$\frac{\Gamma \vdash p \colon \Sigma(x\colon\ \mathbf{type}).t_1? \quad \Gamma \vdash t_0 \colon\ \mathbf{type}}{\Gamma \vdash p_0\, t_0 \colon [x\colon\ = t_0]t_1?}$$
$$\frac{\Gamma,\, x\colon\ \mathbf{type} \vdash p \colon t?}{\Gamma \vdash \mathbf{M}(x\colon\ \mathbf{type}).p \colon \Sigma(x\colon\ \mathbf{type}).t?}$$

I can't figure out satisfaction at all.

$$\frac{v \models [x := t]p\ [\sigma]}{v \models (\mathbf{M}(x\colon\ \mathbf{type}).p)\, t\ [\sigma]}$$
$$\frac{v \models p\ [\sigma,\, t \models x]}{\langle x := t, v\rangle \models \mathbf{M}(x\colon\ \mathbf{type}).p\ [\sigma]}$$