Substructural Types with Class

J. Garrett Morris

Why substructural types?

Short version: state!

- Session types: protocol captured in types

  `send :: (Ch !`$t.s$`, `s`) → IO (Ch `s`)`

- Type-changing update

  `put :: Ref `t` → `u` → IO (Ref `u`)`

- Destructive update

  `update :: Ix `n` → `t` → Array `n` `t` → Array `n` `t

Existing approaches

- Explicit unlimited modality
  - E.g., Wadler, "Linear types can change the world!"
  - Syntactic overhead for unlimited types (`let!`)
- Type modifiers (qualifiers)
  - E.g., Walker, "Substructural type systems"
  - Unexpected types (`lin bool`, `unl (!t.s)`)
  - Code multiplication
- Kinds and subkinding
  - E.g., Mazurak et al., "Lightweight Linear Types…"
  - Code multiplication

This work

Objectives:

- Integrate substructural and unlimited types
  - No syntactic overhead (e.g., `let!`)
  - Partition types (no substructural Booleans or unlimited `!t.s`)
- Avoid code duplication
  - Particularly relevant for higher-order functions

This work

SOL: a functional substructural language

- With principal types and type inference
- Supporting existing functional idioms
- Reduction provably respects substructurality

Hurdles: implicit overloading of

- Duplication/discarding
- Application
- Abstraction

The first hurdle

Here's an innocuous piece of code:

```
twice x = x + x
```

What do we know about the type of *x*:

- It must be numeric (i.e., support +)
- It must be unlimited (i.e., support duplication)

The first hurdle

Characterize unlimited types with a type class

```
class Unl t where
   drop :: t → ()
   dup  :: t → (t, t)
```

The first hurdle

Rewrite twice to use Unl:

```
twice :: (Num t, Unl t) ⇒ t → t
twice x = y + z
   where (y, z) = dup x
```

Really want use of dup and drop to be implicit:

```
twice :: (Num t, Unl t) ⇒ t → t
twice x = x + x
```

Qualified types

System of qualified types in one slide

$$\begin{aligned} \tau, \upsilon, \varphi &::= t \mid K \mid \tau \to \upsilon \\ \pi &::= \ldots \\ \rho &::= \tau \mid \pi \Rightarrow \rho \\ \sigma &::= \rho \mid \forall t.\sigma \end{aligned}$$

$$\frac{P \supseteq Q}{\vdash P \Rightarrow Q} \qquad \frac{\bigwedge\{\vdash P \Rightarrow \pi \mid \pi \in Q\}}{\vdash P \Rightarrow Q}$$

$$\ldots$$

$$\frac{P, \pi \mid \Gamma \vdash M : \rho}{P \mid \Gamma \vdash M : \pi \Rightarrow \rho} \qquad \frac{P \mid \Gamma \vdash M : \pi \Rightarrow \rho \quad \vdash P \Rightarrow \pi}{P \mid \Gamma \vdash M : \rho}$$

$$\frac{x : \sigma \in \Gamma}{P \mid \Gamma \vdash x : \sigma} \qquad \frac{P \mid \Gamma \vdash M : \tau \to \upsilon \quad P \mid \Gamma \vdash N : \tau}{P \mid \Gamma \vdash MN : \upsilon}$$

Substructural qualified types

Substructural variant:

$$\begin{aligned}
\tau, \upsilon, \varphi &::= t \mid K \mid \tau \to \upsilon \mid \tau \multimap \upsilon \mid \tau \oplus \upsilon \\
\pi &::= \texttt{Unl}\,\tau \mid (\rightsquigarrow)\tau \mid \tau \geq \upsilon \\
\rho &::= \tau \mid \pi \Rightarrow \rho \\
\sigma &::= \rho \mid \forall t.\sigma
\end{aligned}$$

$$\frac{P \supseteq Q}{\vdash P \Rightarrow Q} \qquad \frac{\bigwedge\{\vdash P \Rightarrow \pi \mid \pi \in Q\}}{\vdash P \Rightarrow Q}$$

$$\cdots$$

$$\frac{P, \pi \mid H \vdash M : \rho}{P \mid H \vdash M : \pi \Rightarrow \rho} \qquad \frac{P \mid H \vdash M : \pi \Rightarrow \rho \quad \vdash P \Rightarrow \pi}{P \mid H \vdash M : \rho}$$

$$\frac{}{P \mid x : \sigma \vdash x : \sigma} \qquad \frac{P \mid H \vdash M : \tau \to \upsilon \quad P \mid H' \vdash N : \tau}{P \mid H, H' \vdash MN : \upsilon}$$

The first hurdle

Contraction and weakening in terms of `Unl`:

$$\frac{P \mid H, x{:}\,\sigma, x{:}\,\sigma \vdash M{:}\,\sigma' \qquad P \vdash \sigma\ \mathrm{unl}}{P \mid H, x{:}\,\sigma \vdash M{:}\,\sigma'} \qquad \frac{P \mid \mathrm{H} \vdash M{:}\,\sigma' \quad P \vdash \sigma\ \mathrm{unl}}{P \mid H, x{:}\,\sigma \vdash M{:}\,\sigma'}$$

Lifting `Unl`:

$$\frac{\vdash P \Rightarrow \mathtt{Unl}\ \tau}{P \vdash \tau\ \mathrm{unl}} \qquad \frac{P, \pi \vdash \rho\ \mathrm{unl}}{P \vdash \pi \Rightarrow \rho\ \mathrm{unl}} \qquad \frac{P, \mathtt{Unl}\ t \vdash \sigma\ \mathrm{unl}}{P \vdash \forall t.\, \sigma\ \mathrm{unl}}$$

Unlimited types

When is a pair (t, u) unlimited?

```
instance (Unl t, Unl u) ⇒ Unl (t, u) where
    dup (x,y) = ((x,x'), (y,y'))
        where (x,x') = dup x
              (y,y') = dup y
    drop (x,y) = drop y
        where () = drop x
```

Unlimited types

When is a sum (`Either` t u) unlimited?

```
instance (Unl t, Unl u) ⇒ Unl (Either t u) where
    dup (Left x) = (Left x, Left y)
        where (x,y) = dup x
    dup (Right x) = (Right x, Right y)
        where (x,y) = dup x
    drop …
```

Unlimited types

Rules for `Unl` predicates

$$\frac{\vdash P \Rightarrow \texttt{Unl}\ \tau \quad \vdash P \Rightarrow \texttt{Unl}\ \upsilon}{\vdash P \Rightarrow \texttt{Unl}\ (\tau, \upsilon)} \qquad \frac{\vdash P \Rightarrow \texttt{Unl}\ \tau \quad \vdash P \Rightarrow \texttt{Unl}\ \upsilon}{\vdash P \Rightarrow \texttt{Unl}\ (\tau \oplus \upsilon)}$$

What about functions?

When is a function from t to u unlimited?

- Intuition: when the captured environment contains only unlimited values
- Not observable from t and u

Consequence: need distinct function types

$$\frac{}{\vdash P \Rightarrow \mathtt{Unl}\ (\tau \to \upsilon)} \qquad \nvdash \mathtt{Unl}\ (\tau \multimap \upsilon)$$

The second hurdle

Multiple function types make everything worse

```
app (f, x) = f x
```

can have either of the types

```
app :: (t → u, t) → u
app :: (t ⊸ u, t) → u
```

Problem: overloading of juxtaposition

The second hurdle

Characterize function-like things:

```
class (↝) f where
    app :: (f a b, a) → b

instance (↝) (→) where …
instance (↝) (⊸) where …
— no other instances of (↝)
```

The second hurdle

Characterize function-like things:

```
class (↝) f where
    app :: (f a b, a) → b
```

Syntactic sugar:

$$t \leadsto u \equiv t \overset{f}{\leadsto} u \equiv (\leadsto)\ f \Rightarrow f\ t\ u$$

The second hurdle:

Application overloaded for function types:

$$\frac{P \mid H \vdash M : \varphi\tau\upsilon \quad P \mid H' \vdash N : \tau \quad \vdash P \Rightarrow (\leadsto)\varphi}{P \mid H, H' \vdash M\ N : \upsilon}$$

$$\frac{}{\vdash P \Rightarrow (\leadsto)(\to)} \quad \frac{}{\vdash P \Rightarrow (\leadsto)(\multimap)}$$

The third hurdle

Why was app uncurried?

```
app' f x = f x
```

can have any of the types

```
app' :: (t → u) → t → u
app' :: (t → u) → t ⊸ u
app' :: (t ⊸ u) → t ⊸ u
```

The third hurdle

Why was app uncurried?

```
app' f x = f x
```

can use ↝ to abstract argument type

```
app' :: (t ↝ u) → t ⊸ u
```

but result type is still too linear

The third hurdle

Express result linearity as function of argument linearity:

$$\texttt{app'} :: (t \overset{f}{\rightsquigarrow} u) \to t \overset{f}{\rightsquigarrow} u$$
$$\texttt{app'} = \backslash f \to \backslash x \to f\ x$$

- Linearity of a λ-term depends on its captured environment
- Doesn't capture $(t \to u) \to t \multimap u$

The third hurdle

Express result linearity in relation to argument linearity:

$$\texttt{app'} :: f \geq g \Rightarrow (t \overset{f}{\rightsquigarrow} u) \rightarrow t \overset{g}{\rightsquigarrow} u$$
$$\texttt{app'}\ f\ x = f\ x$$

- Unlimited ≥ linear

The third hurdle

Linearity relationships need not be among function types

$$\mathrm{p} :: (t \geq f) \Rightarrow t \to (u \overset{f}{\leadsto} (t,u))$$
$$\mathrm{p}\ x\ y = (x,y)$$

The third hurdle

Linearity of a function depends on its environment:

$$\frac{P \mid H, x{:}\,\tau \vdash M{:}\,\upsilon \quad \vdash P \Rightarrow (\leadsto)\varphi \quad P \vdash H \geq \varphi}{P \mid H_x \vdash \lambda x.\,M{:}\,\varphi\tau\upsilon}$$

Defining ≥

Characterize "more unlimited than" relationship:

```
class t ≥ u
```

Cunning observation: only needs to be defined for function types

```
instance (u → v) ≥ t
instance t ≥ (u ⊸ v)
— no other instances of ≥
```

Defining ≥

Ordering for functions:

$$\frac{}{\vdash (\tau \to \tau') \geq \upsilon} \qquad \frac{}{\vdash \tau \geq (\upsilon \multimap \upsilon')}$$

Lifting:

$$\frac{\vdash P \Rightarrow \tau \geq \varphi}{P \vdash \tau \geq \varphi} \qquad \frac{P, \pi \vdash \rho \geq \varphi}{P \vdash (\pi \Rightarrow \rho) \geq \varphi} \qquad \frac{P, Unl\ t \vdash \sigma \geq \varphi}{P \vdash \forall t. \sigma \geq \varphi}$$

$$\frac{\bigwedge\{P \vdash \sigma \geq \varphi | x{:}\,\sigma \in H\}}{P \vdash H \geq \varphi}$$

SOL syntax-directed typing

Representative rules:

$$\frac{P \vdash^S \Gamma\ \mathrm{unl} \quad (Q \Rightarrow \tau) \sqsubseteq \sigma \quad \vdash P \Rightarrow Q}{P \mid \Gamma, x{:}\,\sigma \vdash^S x{:}\,\tau}$$

$$\frac{P \mid \Gamma, \Delta \vdash^S M{:}\,\varphi\tau\upsilon \quad \vdash P \Rightarrow (\sim)\varphi \quad P \mid \Gamma, \Delta' \vdash^S N{:}\,\tau \quad P \vdash \Gamma\ \mathrm{unl}}{P \mid \Gamma, \Delta, \Delta' \vdash^S MN{:}\,\upsilon}$$

Representative results:

- If $P \mid \Gamma \vdash^S M{:}\,\tau$ and $P \vdash H \approx \Gamma$, then $P \mid H \vdash M{:}\,\tau$
- If $P \mid H \vdash M{:}\,\sigma$ and $P \vdash H \approx \Gamma$, then $Q \mid \Gamma \vdash^S M{:}\,\tau$ where $(P|\sigma) \sqsubseteq Gen(\Gamma, Q \Rightarrow \tau)$.

SOL type inference

Representative rules:

$$\mathcal{M}(S;\Gamma;x;\tau) = UP;U\circ S;\{x\}$$
where $\left(x\!:\forall\vec{t}.P\Rightarrow v\right)\in \mathrm{S}\Gamma$
and $U = \mathrm{Unify}(v[t_i \coloneqq u_i],\tau)$

$$\mathcal{M}(S;\Gamma;\lambda x.M;\tau) = TQ;T;\Sigma\setminus x$$
where $P;T;\Sigma = \mathcal{M}(\mathrm{Unify}(\tau,u_1u_2u_3)\circ S;\Gamma,x\!:u_2;M;u_3)$
and $Q = \{(\rightsquigarrow)u_1\}\cup \mathrm{Leq}(u_1,\Gamma|_\Sigma)\cup \mathrm{Weaken}(x,u_2,\Sigma)$

$$Leq(\varphi,\Gamma) = \{\tau\geq\phi\!:(x\!:\tau)\in\Gamma_\Sigma\}$$
$$Weaken(x,\tau,\Sigma) = \{Unl\ \tau\}\quad \text{if}\quad x\notin\Sigma$$

SOL type inference

Representative results:

- If $\mathcal{M}(S; \Gamma; M; \tau) = P; \mathrm{T}; \Sigma$,

  then $TP \mid \mathrm{T}(\mathrm{S}\Gamma) \vdash^{S} M : T(S\tau)$.

- If $SP \mid S\Gamma \vdash^{S} M : \tau$,

  then $\mathcal{M}(\varepsilon; \Gamma; M; t) = Q; T; \Sigma$ such that:

  - $S = S' \circ T$
  - $\vdash SP \Rightarrow S'(TQ)$
  - $\tau = S'(Tt)$

Principal types for SOL

If $P_0|H \vdash M{:}\,\sigma_0$ and $P_1|H \vdash M{:}\,\sigma_1$, then there is some σ such that

- $\emptyset|H \vdash M{:}\,\sigma$
- $(P_0|\sigma_0) \sqsubseteq \sigma$
- $(P_1|\sigma_1) \sqsubseteq \sigma$.

Continuing work

- Relevant/affine type systems
  - Extended definition of ≥, more arrows
  - Otherwise, should extend directly
- Bounded linearity
  - Related: fractional permissions
  - Problem: arrows
  - Subset where inference is possible?
- Implementation