Parametricity

February 2018

Parametricity

- Polymorphism allows a single piece of code to be instantiated with multiple types.
- Polymorphism is *parametric* when all of the instances behave *uniformly*.
- Where abstraction hides details about an implementation from the outside world, parametricity hides details about the outside world from an implementation.

Parametricity in OCaml

Universal types in OCaml

Universal types in OCaml

```
(* ∀α.α → α *)
let f x = x
```

Universal types in OCaml

```
(* (∀α.List α → Int) → Int *)
let g h = h [1; 2; 3] + h [1.0; 2.0; 3.0]

Characters 27-30:
  let g h = h [1; 2; 3] + h [1.0; 2.0; 3.0]
                                ^^^
Error: This expression has type float
       but an expression was expected of type int
```

Universal types in OCaml

```
Λα::*.λf:α → Int.λx:α.λy:α.
        plus (f x) (f y)
```

```
Λα::*.Λβ::*.λf:∀γ.γ → Int.λx:α.λy:β.
        plus (f [α] x) (f [β] y)
```

Universal types in OCaml

$\Lambda\alpha::*.\lambda$f$:\alpha \to$ Int$.\lambda$x$:\alpha.\lambda$y$:\alpha.$
plus (f x) (f y)

$\Lambda\alpha::*.\Lambda\beta::*.\lambda$f$:\forall\gamma.\gamma \to$ Int$.\lambda$x$:\alpha.\lambda$y$:\beta.$
plus (f [α] x) (f [β] y)

Universal types in OCaml

```
fun f x y -> f x + f y
```

$\forall \alpha :: * .\ (\alpha \to \mathsf{Int}) \to \alpha \to \alpha \to \mathsf{Int}$

$\forall \alpha :: * . \forall \beta :: * .\ (\forall \gamma :: * .\ \gamma \to \mathsf{Int}) \to \alpha \to \beta \to \mathsf{Int}$

Universal types in OCaml

```
(* {h : ∀α.List α → Int} *)
(* ∀α.List α → Int *)
type t = { h : 'a. 'a list -> int }

let len = {h = List.length}

(* t → Int *)
(* (∀α.List α → Int) → Int *)
let g r = r.h [1; 2; 3] + r.h [1.0; 2.0; 3.0]
```

Higher-kinded polymorphism

$$\mathsf{f} \;:\; \forall \mathsf{F} :: * \to * .\, \forall \alpha :: * .\;\; \mathsf{F}\,\alpha \;\to\; (\mathsf{F}\,\alpha \;\to\; \alpha) \;\to\; \alpha$$

$$\mathsf{x} \;:\; \mathsf{List}\;(\mathsf{Int} \times \mathsf{Int})$$

$$\mathsf{f}\;\mathsf{x}$$

Higher-kinded polymorphism

$$F\ \alpha \quad \sim \quad \texttt{List}(\texttt{Int} \times \texttt{Int})$$

$$F = \texttt{List} \qquad \alpha = \texttt{Int} \times \texttt{Int}$$

$$F = \Lambda\beta.\texttt{List}(\beta \times \beta) \qquad \alpha = \texttt{Int}$$

$$F = \Lambda\beta.\texttt{List}(\texttt{Int} \times \texttt{Int})$$

Lightweight higher-kinded polymorphism

A set $\mathbf{F}$ of functions such that:

$$\forall F, G \in \mathbf{F}. \quad F \neq G \quad \Rightarrow \quad \forall t. F(t) \neq G(t)$$

Lightweight higher-kinded polymorphism

```
type 'a t = ('a * 'a) list
```

Lightweight higher-kinded polymorphism

```
type lst = List
type opt = Option

type ('a, 'f) app =
  | Lst : 'a list -> ('a, lst) app
  | Opt : 'a option -> ('a, opt) app
```

```
('a, lst) app ≈ 'a list
```

```
('a, opt) app ≈ 'a option
```

Lightweight higher-kinded polymorphism

```
type 'f map = {
  map: 'a 'b. ('a -> 'b) ->
          ('a, 'f) app -> ('b, 'f) app;
}

let f : 'b map ->
          (int, 'b) app -> (string, 'b) app =
  fun m c ->
    m.map
      (fun x -> "Int: " ^ (string_of_int x))
       c
```

Lightweight higher-kinded polymorphism

```
let lmap : lst map =
  {map = fun f (Lst l) -> Lst (List.map f l)}

let l = f lmap (Lst [1; 2; 3])

let omap : opt map =
  {map = fun f (Opt o) -> Opt (Option.map f o)}

let o = f omap (Opt (Some 6))
```

Lightweight higher-kinded polymorphism

Generalised in the *Higher* library

Functors

```
module type Eq = sig
  type t
  val equal : t -> t -> bool
end

module type SetS = sig
  type t
  type elt
  val empty : t
  val is_empty : t -> bool
  val mem : elt -> t -> bool
  val add : elt -> t -> t
  val remove : elt -> t -> t
  val to_list : t -> elt list
end
```

```
SetS with type elt = foo
```

expands to

```
sig
  type t
  type elt = foo
  val empty : t
  val is_empty : t -> bool
  val mem : elt -> t -> bool
  val add : elt -> t -> t
  val remove : elt -> t -> t
  val to_list : t -> elt list
end
```

```
SetS with type elt := foo
```

expands to

```
sig
  type t
  val empty : t
  val is_empty : t -> bool
  val mem : foo -> t -> bool
  val add : foo -> t -> t
  val remove : foo -> t -> t
  val to_list : t -> foo list
end
```

```
module Set (E : Eq)
  : SetS with type elt := E.t = struct

  type t = E.t list

  let empty = []

  let is_empty = function
    | [] -> true
    | _ -> false

  let rec mem x = function
    | [] -> false
    | y :: rest ->
        if (E.equal x y) then true
        else mem x rest
```

```
  let add x t =
    if (mem x t) then t
    else x :: t

  let rec remove x = function
    | [] -> []
    | y :: rest ->
        if (E.equal x y) then rest
        else y :: (remove x rest)

  let to_list t = t

end
```

```
module IntEq = struct
  type t = int
  let equal (x : int) (y : int) =
    x = y
end

module IntSet = Set(IntEq)
```

Parametricity in System Fω

$$
\begin{aligned}
&\mathsf{SetImpl} = \\
&\quad \lambda\gamma::*.\,\lambda\alpha::*. \\
&\qquad \alpha \\
&\qquad \times\ (\alpha \to \mathsf{Bool}) \\
&\qquad \times\ (\gamma \to \alpha \to \mathsf{Bool}) \\
&\qquad \times\ (\gamma \to \alpha \to \alpha) \\
&\qquad \times\ (\gamma \to \alpha \to \alpha) \\
&\qquad \times\ (\alpha \to \mathsf{List}\ \gamma)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{empty} = \Lambda\gamma::*.\,\Lambda\alpha::*.\,\lambda s : \mathsf{SetImpl}\ \gamma\ \alpha.\pi_1\ s \\
&\mathsf{is\_empty} = \Lambda\gamma::*.\,\Lambda\alpha::*.\,\lambda s : \mathsf{SetImpl}\ \gamma\ \alpha.\pi_2\ s \\
&\mathsf{mem} = \Lambda\gamma::*.\,\Lambda\alpha::*.\,\lambda s : \mathsf{SetImpl}\ \gamma\ \alpha.\pi_3\ s \\
&\mathsf{add} = \Lambda\gamma::*.\,\Lambda\alpha::*.\,\lambda s : \mathsf{SetImpl}\ \gamma\ \alpha.\pi_4\ s \\
&\mathsf{remove} = \Lambda\gamma::*.\,\Lambda\alpha::*.\,\lambda s : \mathsf{SetImpl}\ \gamma\ \alpha.\pi_5\ s \\
&\mathsf{to\_list} = \Lambda\gamma::*.\,\Lambda\alpha::*.\,\lambda s : \mathsf{SetImpl}\ \gamma\ \alpha.\pi_6\ s
\end{aligned}
$$

$$\mathsf{EqImpl} = \lambda\gamma :: * . \gamma \to \gamma \to \mathsf{Bool}$$

$$\mathsf{equal} = \Lambda\gamma :: * . \lambda \mathsf{s} : \mathsf{EqImpl}\ \gamma . \mathsf{s}$$

```
set_package =
  Λγ::*. λeq:EqImpl γ.
    pack List γ,⟨
         nil [γ],
          isempty [γ],
          λn:γ.fold [γ] [Bool]
            (λx:γ.λy:Bool.or y (equal [γ] eq n x))
            false,
          cons [γ],
          λn:γ.fold [γ] [List γ]
            (λx:γ.λl:List γ.
               if (equal [γ] eq n x) [List γ] l
                  (cons [γ] x l))
            (nil [γ]),
          λl:List γ.l ⟩
    as ∃α::*.SetImpl γ α
```

$$\frac{\Gamma \vdash M : \forall \alpha :: K.A \qquad \Gamma \vdash B :: K}{\Gamma \vdash M\ [B] : A[\alpha := B]}\ \forall\text{-elim}$$

Relational parametricity

Relational parametricity

We can give precise descriptions of parametricity using relations between types.

Relational parametricity

Given a type T with free variables $\alpha, \beta_1, \ldots, \beta_n$:

$$\forall B_1. \ldots \forall B_n. \forall x : (\forall \alpha. T[\alpha, B_1, \ldots, B_n]).$$
$$\forall \gamma.\ \forall \delta.\ \forall \rho \subset \gamma \times \delta.$$
$$[\![T]\!][\rho, =_{B_1}, \ldots, =_{B_n}](x[\gamma],\ x[\delta])$$

Relational parametricity

Any value with a universal type must preserve all relations between any two types that it can be instantiated with.

Theorems for free

Parametricity applied to $\forall\alpha.\alpha \to \alpha$:

$$\begin{aligned}
&\forall f : (\forall\alpha.\alpha \to \alpha).\\
&\quad \forall\gamma.\ \forall\delta.\ \forall\rho \subset \gamma \times \delta.\\
&\quad\quad \forall u : \gamma.\ \forall v : \delta.\\
&\quad\quad\quad \rho(u,\ v) \Rightarrow \rho(f[\gamma]\, u,\ f[\delta]\, v)
\end{aligned}$$

Define a relation is_u to represent being equal to a value $u : T$:

$$\mathsf{is}_u(x : T, y : T) \quad = \quad (x =_T u) \ \wedge \ (y =_T u)$$

Theorems for free

$$\forall f : (\forall \alpha.\alpha \to \alpha).$$
$$\forall \gamma.\forall u : \gamma.$$
$$\mathsf{is}_u(u, u) \Rightarrow \mathsf{is}_u(f[\gamma]\, u,\ f[\gamma]\, u)$$

$$\forall f : (\forall \alpha.\alpha \to \alpha).$$
$$\forall \gamma.\forall u : \gamma.$$
$$f[\gamma]\, u \;=_\gamma u$$

Parametricity applied to $\forall\alpha.\mathsf{List}\,\alpha \to \mathsf{List}\,\alpha$:

$$\forall f : (\forall\alpha.\mathsf{List}\,\alpha \to \mathsf{List}\,\alpha).$$
$$\forall\gamma.\ \forall\delta.\ \forall\rho \subset \gamma \times \delta.$$
$$\forall u : \mathsf{List}\,\gamma.\ \forall v : \mathsf{List}\,\delta.$$
$$[\![\mathsf{List}\,\alpha]\!][\rho](u,\ v) \Rightarrow [\![\mathsf{List}\,\alpha]\!][\rho](f[\gamma]\,u,\ f[\delta]\,v)$$

The System F encoding for lists:

$$\mathsf{List}\ \alpha = \forall\beta.\, \beta \to (\alpha \to \beta \to \beta) \to \beta$$

$$\mathsf{nil}_\alpha = \Lambda\beta.\, \lambda \mathsf{n} : \beta.\, \lambda \mathsf{c} : \alpha \to \beta \to \beta.\ \mathsf{n}$$

$$\begin{aligned}\mathsf{cons}_\alpha = \lambda \mathsf{x} : \alpha.\, \lambda \mathsf{xs} : \mathsf{List}\ \alpha.\\ \Lambda\beta.\, \lambda \mathsf{n} : \beta.\, \lambda \mathsf{c} : \alpha \to \beta \to \beta.\\ \mathsf{c}\ \mathsf{x}\ (\mathsf{xs}\ [\beta]\ \mathsf{n}\ \mathsf{c})\end{aligned}$$

The relational interpretation of the System F encoding for lists:

$$
\begin{aligned}
&[\![\mathsf{List}\,\alpha]\!][\rho] = \\
&\qquad (x : \mathsf{List}A,\ y : \mathsf{List}B). \\
&\qquad\quad \forall\gamma.\ \forall\delta.\ \forall\rho' \subset \gamma \times \delta. \\
&\qquad\qquad \forall n : \gamma.\ \forall m : \delta. \\
&\qquad\qquad\quad \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
&\qquad\qquad\qquad \rho'(n,\ m) \Rightarrow \\
&\qquad\qquad\qquad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
&\qquad\qquad\qquad\qquad \rho(a,b) \Rightarrow \rho'(u,v) \Rightarrow \rho'(c\,a\,u,\ d\,b\,v)) \Rightarrow \\
&\qquad\qquad\qquad\qquad \rho'(x[\gamma]\,n\,c,\ y[\delta]\,m\,d)
\end{aligned}
$$

If $x = nil_A$ and $y = nil_B$:

$$\begin{array}{l}
\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta. \\
\quad \forall n : \gamma.\ \forall m : \delta. \\
\quad\ \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
\quad\ \forall a : A. \forall u : \gamma.\ \forall b : B. \forall v : \gamma. \\
\quad\quad \rho'(n,\ m) \Rightarrow \\
\quad\quad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
\quad\quad\quad\quad \rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow \\
\quad\quad\quad\quad \rho'(nil_A[\gamma]\, n\, c,\ nil_B[\delta]\, m\, d)
\end{array}$$

If $x = nil_A$ and $y = nil_B$:

$$\begin{array}{l}
\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta. \\
\quad \forall n : \gamma.\ \forall m : \delta. \\
\quad\ \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
\quad\ \forall a : A. \forall u : \gamma.\ \forall b : B. \forall v : \gamma. \\
\quad\quad \rho'(n,\ m) \Rightarrow \\
\quad\quad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
\quad\quad\quad\quad \rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow \\
\quad\quad\quad\quad \rho'(n,\ m)
\end{array}$$

If $x = nil_A$ and $y = nil_B$:

$$
\begin{array}{l}
\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta. \\
\quad \forall n : \gamma.\ \forall m : \delta. \\
\quad\ \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
\quad\ \forall a : A. \forall u : \gamma.\ \forall b : B. \forall v : \gamma. \\
\quad\quad \rho'(n,\ m) \Rightarrow \\
\quad\quad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
\quad\quad\quad\quad \rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow \\
\quad\quad\quad\quad \rho'(n,\ m)
\end{array}
$$

If $x = cons_A\, i\, l$ and $y = cons_B\, j\, k$:

$$\begin{array}{l}
\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta. \\
\quad \forall n : \gamma.\ \forall m : \delta. \\
\quad\quad \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
\quad\quad\quad \rho'(n,\ m) \Rightarrow \\
\quad\quad\quad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
\quad\quad\quad\quad\quad \rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow \\
\quad\quad\quad\quad\quad\quad \rho'(cons_A[\gamma]\, i\, l\, n\, c,\ cons_B[\delta]\, j\, k\, m\, d)
\end{array}$$

If $x = cons_A\, i\, l$ and $y = cons_B\, j\, k$:

$$
\begin{array}{l}
\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta. \\
\quad \forall n : \gamma.\ \forall m : \delta. \\
\quad\ \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
\quad\quad \rho'(n,\ m) \Rightarrow \\
\quad\quad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
\quad\quad\quad\quad \rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow \\
\quad\quad\quad\quad \rho'(c\, i\, (l[\gamma]\, n\, c),\ d\, j\, (k[\gamma]\, m\, d))
\end{array}
$$

If $x = cons_A\, i\, l$ and $y = cons_B\, j\, k$:

$$\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta.$$
$$\forall n : \gamma.\ \forall m : \delta.$$
$$\forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta.$$
$$\rho'(n,\ m) \Rightarrow$$
$$(\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta.$$
$$\rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow$$
$$\rho'(c\, i\, (l[\gamma]\, n\, c),\ d\, j\, (k[\gamma]\, m\, d))$$

If $x = cons_A\, i\, l$ and $y = cons_B\, j\, k$:

$$\begin{array}{l}
\forall \gamma.\ \forall \delta.\ \forall \rho' \subset \gamma \times \delta. \\
\quad \forall n : \gamma.\ \forall m : \delta. \\
\quad\quad \forall c : A \to \gamma \to \gamma.\ \forall d : B \to \delta \to \delta. \\
\quad\quad\quad \rho'(n,\ m) \Rightarrow \\
\quad\quad\quad\quad (\forall a : A.\ \forall b : B.\ \forall u : \gamma.\ \forall v : \delta. \\
\quad\quad\quad\quad\quad \rho(a, b) \Rightarrow \rho'(u, v) \Rightarrow \rho'(c\, a\, u,\ d\, b\, v)) \Rightarrow \\
\quad\quad\quad\quad\quad \rho(i, j)\ \wedge\ \rho'(l[\gamma]\, n\, c,\ k[\gamma]\, m\, d)
\end{array}$$

The relational interpretation of the System F encoding for lists:

$$[\![\mathsf{List}\,\alpha]\!][\rho](x : \mathsf{List}\,A,\ y : \mathsf{List}\,B) =$$

$$\begin{cases} \rho(i,j)\ \wedge\ [\![\mathsf{List}\,\alpha]\!][\rho](l,k), & x = cons_{\,A}\,i\,l \wedge y = cons_{\,B}\,j\,k \\ true, & x = nil_{\,A} \wedge y = nil_{\,B} \\ false, & otherwise \end{cases}$$

Define a relation $\langle g \rangle$ to represent a function $g : A \to B$

$$\langle g \rangle (x : A, y : B) \quad = \quad (g\, x =_B y)$$

Apply the relational interpretation for lists to $\langle g \rangle$:

$$[\![\mathsf{List}\,\alpha]\!][\langle g \rangle](x : \mathsf{List}\,A,\ y : \mathsf{List}\,B) =$$

$$\begin{cases} g\,i =_B j \ \wedge\ [\![\mathsf{List}\,\alpha]\!][\langle g \rangle](l,k), & x = cons_A\,i\,l \wedge y = cons_B\,j\,k \\ true, & x = nil_A \wedge y = nil_B \\ false, & otherwise \end{cases}$$

Apply the relational interpretation for lists to $\langle g \rangle$:

$$[\![\mathsf{List}\,\alpha]\!][\langle g \rangle](xs : \mathsf{List}\,A, ys : \mathsf{List}\,B) =$$
$$\mathsf{map}[A][B]\,g\,xs =_{\mathsf{List}\,B} ys$$

A free theorem for $\forall\alpha.\mathsf{List}\,\alpha \to \mathsf{List}\,\alpha$:

$$\begin{aligned}
&\forall f : (\forall\alpha.\mathsf{List}\,\alpha \to \mathsf{List}\,\alpha).\\
&\quad \forall\gamma.\ \forall\delta.\ \forall g : \gamma \to \delta\\
&\qquad \forall u : \mathsf{List}\,\gamma.\ \forall v : \mathsf{List}\,\delta.\\
&\qquad\quad \mathsf{map}[\gamma][\delta]\,g\,(f[\gamma]\,u) = f[\delta]\,(\mathsf{map}[\gamma][\delta]\,g\,u)
\end{aligned}$$

Terms and conditions apply

Terms and conditions apply

```
let f (x : 'a) : 'a =
  Printf.printf "Launch missiles\n";
  x

let f (x : 'a) : 'a = raise Exit

let rec f (x : 'a) : 'a = f x
```

Parametricity applied to $\forall\alpha.\alpha \to \alpha \to \mathsf{Bool}$:

$$\forall f : (\forall\alpha.\alpha \to \alpha \to \mathsf{Bool}).$$
$$\forall\gamma.\ \forall\delta.\ \forall\rho \subset \gamma \times \delta.$$
$$\forall u : \gamma.\ \forall v : \delta.\ \forall u' : \gamma.\ \forall v' : \delta.$$
$$\rho(u,\ v) \Rightarrow \rho(u',\ v') \Rightarrow$$
$$[\![Bool]\!][\rho](f[\gamma]\,u\,u',\ f[\delta]\,v\,v')$$

Parametricity applied to $\forall\alpha.\alpha \to \alpha \to \mathsf{Bool}$:

$$\begin{aligned}
&\forall f : (\forall\alpha.\alpha \to \alpha \to \mathsf{Bool}).\\
&\quad \forall\gamma.\ \forall\delta.\ \forall\rho \subset \gamma \times \delta.\\
&\qquad \forall u : \gamma.\ \forall v : \delta.\ \forall u' : \gamma.\ \forall v' : \delta.\\
&\qquad\qquad \rho(u,\ v) \Rightarrow \rho(u',\ v') \Rightarrow\\
&\qquad\qquad\qquad (f[\gamma]\,u\,u' =_{Bool} f[\delta]\,v\,v')
\end{aligned}$$

$$\forall f : (\forall \alpha.\alpha \to \alpha \to \mathsf{Bool}).$$
$$\forall \gamma.\ \forall \delta.$$
$$\forall u : \gamma.\ \forall v : \delta.\ \forall u' : \gamma.\ \forall v' : \delta.$$
$$(f[\gamma]\, u\, u' =_{Bool} f[\delta]\, v\, v')$$

```
val (=) : 'a -> 'a -> bool
```

Parametricity in geometery and mechanics

- Relational parametricity gives a good account of invariance under change.
- The use of invariance under change to derive useful consequences is a technique much older than programming languages.
- Stepping beyond OCaml and System Fω we can apply parametricity to other areas of mathematics.

Parametricity in geometery and mechanics

Consider representing points as pairs of real numbers:

```
type point = real * real
```

The meaning of these real numbers depends on their choice of origin.

We add a new kind T_2 to represent origins:

$$\frac{}{T_2 \text{ is a kind}}$$

We define type-level operations for T_2 corresponding to those of an abelian group:

$$\frac{}{\Gamma \vdash 0 :: T_2}$$

$$\frac{\Gamma \vdash A :: T_2 \qquad \Gamma \vdash B :: T_2}{\Gamma \vdash A + B :: T_2}$$

etc.

Now we can index the point type by its choice of origin:

```
type 't point
```

and define various vector operations on them:

```
val (+) : 't point -> 's point -> ('t + 's) point
val (-) : 't point -> 's point -> ('t - 's) point
val cross : 0 point -> 0 point -> real
```

The following function gives the area of a triangle represented by three points:

```
let area a b c =
  (abs (cross (c - a) (b - a))) / 2
```

it has type:

```
val area: 't point -> 't point -> 't point -> real
```

Parametricity in geometery and mechanics

From relational parametricity, the type:

```
't point -> 't point -> 't point -> real
```

gives a free theorem that the area of the triangle is invariant under translation.

Parametricity in geometery and mechanics

By further enriching our type system with kinds for other mathematical objects, we can take this even further.

Parametricity in geometery and mechanics

The following equation is a Lagrangian function describing a mechanical system containing two particals of mass m connected by a spring with spring constant k and constrained to move in one-dimension:

$$L(t, x_1, x_2, \dot{x_1}, \dot{x_2}) = \frac{1}{2}m(\dot{x_1}^2 + \dot{x_2}^2) - \frac{1}{2}k(x_1 - x_2)^2$$

Parametricity in geometery and mechanics

It can be given a type like this:

$$\forall y : \mathbf{T}(1).$$
$$C^{\infty}(\mathbb{R}(1,0) \times \mathbb{R}(1,y) \times \mathbb{R}(1,y) \times \mathbb{R}(1,0) \times \mathbb{R}(1,0),\ \mathbb{R}(1,0))$$

- Using relational parametricity, we can derive as a free theorem of this type that the system is invariant to spatial translation.
- By Noether's theorem, this demonstrates that the system conserves linear momentum.

Next time

$$\frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

Next time

$$\Gamma \vdash A$$