Parametricity

Leo White

Jane Street

February 2016

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

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

$$\Lambda\alpha\texttt{::}*\texttt{.}\Lambda\beta\texttt{::}*\texttt{.}\lambda\texttt{f:}\forall\gamma\texttt{.}\gamma \to \texttt{Int.}\lambda\texttt{x:}\alpha\texttt{.}\lambda\texttt{y:}\beta\texttt{.}$$
$$\qquad\texttt{plus (f [}\alpha\texttt{] x) (f [}\beta\texttt{] y)}$$

Universal types in OCaml

λf .λx .λy .
plus (f x) (f y)

λf .λx .λy .
plus (f x) (f y)

Universal types in OCaml

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

$\forall\alpha$::*. (α $\rightarrow$ Int) $\rightarrow$ α $\rightarrow$ α $\rightarrow$ Int

$\forall\alpha$::*.$\forall\beta$::*. ($\forall\gamma$::*. γ $\rightarrow$ Int) $\rightarrow$ α $\rightarrow$ β $\rightarrow$ Int

Universal types in OCaml

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

let len = {h = List.length}

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

Higher-kinded polymorphism

$\texttt{f : } \forall \texttt{F::*} \rightarrow \texttt{*.} \forall \alpha \texttt{::*. F } \alpha \rightarrow \texttt{(F } \alpha \rightarrow \alpha\texttt{)} \rightarrow \alpha$

$\texttt{x : List (Int} \times \texttt{Int)}$

`f x`

Higher-kinded polymorphism

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

$$\begin{array}{ll} F = \texttt{List} & \alpha = \texttt{Int} \times \texttt{Int} \\ F = \Lambda\beta.\texttt{List}(\beta \times \beta) & \alpha = \texttt{Int} \\ F = \Lambda\beta.\texttt{List}(\texttt{Int} \times \texttt{Int}) & \end{array}$$

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` $\approx$ `'a list`

`('a, opt)app` $\approx$ `'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

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
```

Functors

```
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
```

Functors

```
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
```

Functors

```
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
```

Functors

```
  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
```

Functors

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

module IntSet = Set(IntEq)
```

Parametricity in System Fω

Universal types

```
SetImpl =
  λγ::*.λα::*.
      α
      × (α → Bool)
      × (γ → α → Bool)
      × (γ → α → α)
      × (γ → α → α)
      × (α → List γ)
```

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

Universal types

$$
\begin{array}{l}
\texttt{EqImpl =} \\
\quad \lambda\gamma\texttt{::*.}\gamma \rightarrow \gamma \rightarrow \texttt{Bool}
\end{array}
$$

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

Universal types

```
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
```

Universal types

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

Relational parametricity

Relational parametricity

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

Relational parametricity

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

$$
\begin{aligned}
&\forall \beta_1. \ldots \forall \beta_n. \forall x : (\forall \alpha. T). \\
&\qquad \forall \gamma.\ \forall \delta.\ \forall \rho \subset \gamma \times \delta. \\
&\qquad\qquad T[\rho, =_{\beta_1}, \ldots, =_{\beta_n}](x[\gamma],\ x[\delta])
\end{aligned}
$$

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

$$
\begin{aligned}
&\forall f : (\forall \alpha.\alpha \to \alpha). \\
&\quad \forall \gamma.\forall u : \gamma. \\
&\qquad f[\gamma]\, u \;=_{\gamma}\; u
\end{aligned}
$$

Theorems for free

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

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

Theorems for free

The System F encoding for lists:

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

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

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

Theorems for free

The relational substitution of the System F encoding for lists:

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

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

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

Theorems for free

The relational substitution of the System F encoding for lists:

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

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

Theorems for free

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

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

Theorems for free

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

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

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

Theorems for free

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

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

Theorems for free

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

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

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
```

Terms and conditions apply

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

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

Terms and conditions apply

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

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

Terms and conditions apply

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

Terms and conditions apply

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