

Beluga: Functional Programming with Binders

Ben Lippmeier

University of New South Wales

FP-Syd 2013/02/27

Like Twelf...



(CMU crowd)

- Carsten Schürmann
- Brigitte Pientka
- Roberto Virga
- Kevin Watkins
- Peter Lee
- Frank Pfenning
- Bob Harper
- Dan Licata

Like Twelf...

```
tp      : type.  
unit    : tp.  
arrow   : tp -> tp -> tp.  
  
tm      : type.  
empty   : tm.  
app     : tm -> tm -> tm.  
lam     : tp -> (tm -> tm) -> tm.
```

Like Twelf...

`of` : `tm -> tp -> type.`

`of-empty` : `of empty unit.`

`of-app` : `of (app E1 E2) T`
`<- of E1 (arrow T2 T)`
`<- of E2 T2.`

`of-lam` : `of (lam T2 ([x] E x)) (arrow T2 T)`
`<- ({x: tm} of x T2 -> of (E x) T).`

Like Twelf...

`of` : `tm -> tp -> type.`

`of-empty` : `of empty unit.`

`of-app` : `of (app E1 E2) T`
<- `of E1 (arrow T2 T)`
<- `of E2 T2.`

`of-lam` : `of (lam T2 ([x] E x)) (arrow T2 T)`
<- `{x: tm} of x T2 -> of (E x) T).`

$\text{TE} \vdash E1 :: T2 \rightarrow T \quad \text{TE} \vdash E2 :: T2$

$\text{TE} \vdash E1 E2 :: T$

Like Twelf...

`of` : `tm -> tp -> type.`

`of-empty` : `of empty unit.`

`of-app` : `of (app E1 E2) T`
`<- of E1 (arrow T2 T)`
`<- of E2 T2.`

`of-lam` : `of (lam T2 ([x] E x)) (arrow T2 T)`
`<- ({x: tm} of x T2 -> of (E x) T).`

$$\frac{\text{TE}, x : T2 \vdash E :: T}{\text{TE} \vdash \lambda(x : T2). E :: T2 \rightarrow T}$$

`step` : `tm -> tm -> type`.

`step-app-1`

`: step (app E1 E2) (app E1' E2)`
`<- step E1 E1'`.

`step-app-2`

`: step (app E1 E2) (app E1 E2')`
`<- value E1`
`<- step E2 E2'`.

`step-app-beta`

`: step (app (lam T2 ([x] E x)) E2) (E E2)`
`<- value E2`.

```
pres : step E E' -> of E T -> of E' T -> type.  
%mode pres +Dstep +Dof -Dof'.
```

```
pres-app-1
```

```
: pres (step-app-1 (DstepE1 : step E1 E1'))  
      (of-app (DofE2 : of E2 T2)  
              (DofE1 : of E1 (arrow T2 T)))  
      (of-app DofE2 DofE1')  
<- pres DstepE1 DofE1 (DofE1' : of E1' (arrow T2 T)).
```

```
pres-app-2
```

```
: pres (step-app-2 (DstepE2 : step E2 E2') (DvalE1 : value E1))  
      (of-app (DofE2 : of E2 T2)  
              (DofE1 : of E1 (arrow T2 T)))  
      (of-app DofE2' DofE1)  
<- pres DstepE2 DofE2 (DofE2' : of E2' T2).
```

```
pres-app-beta
```

```
: pres (step-app-beta (Dval : value E2))  
      (of-app (DofE2 : of E2 T2)  
              (of-lam (([x] [dx] DofE x dx)  
                       : {x : tm} {dx : of x T2} of (E x) T)))  
      (DofE E2 DofE2)).
```

```
%worlds () (preserv _ _ _).
```

```
%total D (preserv D _ _).
```


People at McGill University (Montreal, Canada)



Brigitte Pientka



Andreas Abel



Joshua Dunfield



Max Planck
Institute



Mathieu Boespflug



Andrew Cave

Beluga: like Twelf...

`tp` : type. `%name tp T`.

`tm` : type. `%name tm M`.

`arr` : `tp -> tp -> tp`.

`app` : `tm -> tm -> tm`.

`lam` : `tp -> (tm -> tm) -> tm`.

Beluga: like Twelf... but Functional (not Relational)

`has_type` : tm -> tp -> type.

`%name` has_type D.

`is_app` : has_type E1 (arr T1 T2)
-> has_type E2 T1
-> has_type (app E1 E2) T2.

`is_lam` : ({x:tm} has_type x T1 -> has_type (E x) T2)
-> has_type (lam T1 (\x. E x)) (arr T1 T2).

Beluga: like Twelf... but Functional (not Relational)

`has_type` : `tm -> tp -> type`.

`%name` `has_type` `D`.

`is_app` : `has_type E1 (arr T1 T2)`
 `-> has_type E2 T1`
 `-> has_type (app E1 E2) T2`.

`is_lam` : `({x:tm} has_type x T1 -> has_type (E x) T2)`
 `-> has_type (lam T1 (\x. E x)) (arr T1 T2)`.

$\lambda(x : T1) . E$

`step` : `tm -> tm -> type.` `%name step S.`

`s_app1` : `step E1 E1'`
`-> step (app E1 E2) (app E1' E2).`

`s_app2` : `value E1`
`-> step E2 E2'`
`-> step (app E1 E2) (app E1 E2').`

`s_app3` : `value E2`
`-> step (app (lam T (\x. E1 x)) E2) (E1 E2).`

```

rec pres : [. has_type E T] -> [. step E E']
          -> [. has_type E' T]
= fn d => fn s =>
case s of
| [. s_app1 S1] =>
  let [. is_app D1 D2] = d in
  let [. D1']          = pres [. D1] [. S1]
  in  [. is_app D1' D2]

| [. s_app2 V S2] =>
  let [. is_app D1 D2] = d in
  let [. D2']          = pres [. D2] [. S2]
  in  [. is_app D1 D2']

| [. s_app3 V] =>
  let [. is_app (is_lam (\x. (\d. (D1 x d)))) D2] = d
  in  [. (D1 _ D2)]

```

Programming with Contexts

```
tm    : type.  
lam   : (tm -> tm) -> tm.  
app   : tm -> tm -> tm.
```

```
schema ctx = tm;
```

```
datatype Clos : ctype  
= C1 : (g:ctx) [g,x:tm . tm]  
      -> ([g . tm] -> Clos)  
      -> Clos;
```

```

datatype Clos : ctype
  = Cl : (g:ctx) [g,x:tm. tm] -> ([g. tm] -> Clos) -> Clos;

rec reduce : (g:ctx) [g. tm] -> ([g. tm] -> Clos) -> Clos
  = fn e => fn env
    => case e of
      | [g. #p ..]           => env [g. #p ..]

      | [g. lam (\x. E .. x)] => Cl [g, x:tm. E .. x] env

      | [g. app (E1 ..) (E2 ..)]
        => case reduce [g. E1 ..] env of
          | Cl [h, x:tm. E .. x] env'
            => let v = reduce [g. E2 ..] env
              in reduce [h, x:tm. E .. x]
                (fn var => case var of
                  | [h, x : tm. x]           => v
                  | [h, x : tm. #p ..] => env' [h . #p ..]);

```


$\lambda(x : T1) . E$

datatype Clos : ctype

= C1 : (g:ctx) [g,x:tm. tm] -> ([g. tm] -> Clos) -> Clos;

rec reduce : (g:ctx) [g. tm] -> ([g. tm] -> Clos) -> Clos

= **fn** e => **fn** env

=> **case** e **of**

| [g. #p ..] => env [g. #p ..]

| [g. lam (\x. E .. x)] => C1 [g, x:tm. E .. x] env

| [g. app (E1 ..) (E2 ..)]

=> **case** reduce [g. E1 ..] env **of**

| C1 [h, x:tm. E .. x] env'

=> **let** v = reduce [g. E2 ..] env

in reduce [h, x:tm. E .. x]

(**fn** var => **case** var **of**

| [h, x : tm. x] => v

| [h, x : tm. #p ..] => env' [h . #p ..]);

Beluga summary

- Cleaner than Twelf (functional rather than relational).
- Direct support for programming with binders and contexts.
- Looks promising for formalising programming languages.
- They have a coverage checker, but no totality checker yet.
- Diverging expressions can be written in the “proof language.”
- Mentions of automated prover tactics on the web, but no implementation yet.
- Active project, last release 7/2/2013

Competitors



Elf → Twelf



Elphin → Delphin

Agda

Epigram

Idris

Coq

Matita

NuPRL

Isabelle/HOL

Abella (~2010, dead?)

LEGO (~1998 dead)