PolyLib - a polytypic function library

PolyLib — a library of polytypic functions

Patrik Jansson Johan Jeuring

May 15, 1998

1 Introduction

polytypic func-

tions 12 fi

regular[1] P P 8

x H k w

fi P P -

w

D w w P P -

, x fi , , w , -

, T ,

w -k w `cata` `map`, w -k w

`propagate` `thread` W

P P: P L 10, B T -

P L , ,

T w x w

fi W w w

fi

O , L -

k J , D , P H k

x

w - : w -

ffi , w x *write*

O , *use*

, q ,

T fi P P; w x

w O

k

w k w fi

[1] A type constructor `d` is regular if the datatype `d a` contains no function spaces, and if the argument of `d` is the same on the left- and right-hand side of its definition.

I w w -
w , x

2 Polytypic programs

U ff P P ( ) w
x , w
q , w , -
, w
, fi w B
4 T w
, w

2.1 Describing polytypic functions

O ( ) w -
: ; ( ) ;
k w ; k w ; k-
(H w w
4) x :

```
pmap :: (a -> b) -> d a -> d b
```

pmap k f x d a,
f **Also known as**: map 12 , $\texttt{map}_n$ 11 **Known uses**:
E w ! **Background**: T w fi

w -
w *specify* T
fi-
,
T , w w fi ,
w -

2.2 Library overview

W fi , fi 1 T fi
w map, cata ana
T fi
T w z
, q T
T fi (
) , k fl T w

`pmap`, `fmap`, `cata` `ana`, `hylo`, `para` `crush`, `fcrush`	`pzip`, `fzip` `punzip`, `funzip` `pzipWith`, `pzipWith'` `pequal`, `fequal`	`pmapM`, `fmapM`, `cataM` `anaM`, `hyloM`, `paraM` `propagate`, `cross` `thread`, `fthread`
(a) Recursion operators	(b) Zips etc.	(c) Monad op's

`flatten`, `fflatten` `fl_par`, `fl_rec`, `conc`	`psum`, `size`, `prod` `pand`, `pall` `por`, `pany`, `pelem`
(d) Flatten functions	(e) Miscellaneous

1: O w P L

3 PolyLib

H k w
p ( ) H k (T
w f) I w b <- a -
a -> b k (
) fi T
w : $\texttt{pmap}_d$
T w fi d a
I x Regular d => ..., w
w d w fix
f W w f FunctorOf d

3.1 Recursion operators

```
pmap :: (a -> b) -> d a -> d b
fmap :: (a -> c) -> (b -> d) -> f a b -> f c d
```

pmap k f x d a,
f a x W d
, $\texttt{pmap}_d$ -
$\texttt{fmap}_f$ f **Also known as**: map 12 , $\texttt{map}_n$ 11 I **charity** 3 $\texttt{map}_d$ f x w d f (x) **Known uses**: E w ! fmap fi pmap, cata, ana, hylo, para P L **Background**: T w fi w k B M , 2, 16 T map T H k :

```
map    Functor f => (a->b) -> f a -> f b
```

pmap
pmap

H k map

```
cata :: (FunctorOf d a b -> b) -> (d a -> b)
ana  :: (FunctorOf d a b <- b) -> (d a <- b)
hylo :: (f a b -> d) -> (c -> f a b) -> (c -> d)
para :: (d a -> FunctorOf d a b -> b) -> (d a -> b)
```

w d a: T , cata, “ ” w -
T cata w fi FunctorOf q :

```
cata    (f a b -> b) -> (Mu f a -> b)
```

T , ana, w k -
T , hylo, w
, ,
para, cata
,

Also known as:

P L	ML 1	Sq	**charity** 3
cata i	fold_1 i	$\|i\|$	i
ana o	-	o	(o)

cata para V 5 **Known uses**: V fi cata: pmap, crush, thread, flatten, propagate, para re rite **Background**: T , cata, - H k foldr , ana, ( ) C w M 14, 15 -

fi : hylo i o = cata i . ana o T 17 , para, d a I d a

```
crush  :: (a->a->a) -> a -> d a -> a
fcrush :: (a->a->a) -> a -> f a a -> a
```

T crush op e k x op w a x (T

e) **Known uses**: w 3 5
M ff
Background: T fi `crush` 18
`op` w `e, crush op e` fi `foldr op e . flatten`
`crush` `fold`
`cata` `fold`

3.2 Zips

```
pzip   :: (d a d b) -> Maybe ( d (a b) )
punzip :: d (a b) -> (d a d b)
fzip   :: (f a b f c d) -> Maybe ( f (a c) (b d) )
funzip :: f (a c) (b d) -> (f a b f c d)
```

`punzip` k
fi
`pzip` `punzip`: k z
`ust` w ,
`othing` w **Also known as**: $\texttt{zip}_m$ 11 , **zip.×.d** 6 , **Known uses**: `fzip` fi `pzipWith` **Background**: T
`zip`

```
zip     a  ->  b  ->  (a b)
```

w `Maybe`
w (I
,)
(“ ”) `pzip: transpose` (**zip.d.e** 6)

```
transpose    d (e a) -> e (d a)
```

w fi z R 19 fi , H jk
& B k 6

```
pzipWith  :: ((a b) -> Maybe c) -> (d a d b) -> Maybe (d (a b))
pzipWith' :: (FunctorOf d c e -> e) -> ((d a  d b) -> e) ->
             ((a  b) -> c) -> (d a d b) -> e
```

`pzipWith op` w k k `pzip` `op`
w j z
, w `Maybe-`
2

[2] The type constructor `Maybe` can be replaced by any monad with a zero, but we didn't want to clutter up the already complicated type with contexts.

pzipWith' pzipWith w
ff I pzipWith' ins fail op, op
, fail
w w , ins
(T ins fi
cata) **Also known as**: zipop$_m$ 11 **Known uses**: pzipWith'
fi q , fi **Background**:
pzipWith H k zipWith

```
zipWith    (a->b->c) ->  a  ->  b  ->  (a b)
```

pzipWith' w pzip j pzipWith ust

```
pequal :: (a->b->Bool) -> d a -> d b -> Bool
fequal :: (a->b->Bool) -> (c->d->Bool) -> f a c -> f b d -> Bool
```

T x pequal eq x y k x y q
q eq w **Known uses**: fequal fi w w
q **Background**: q
20 pequal
H k Eq- :

```
(==)    Eq a => d a -> d a -> Bool
(==) =  pequal (==)
```

I H k q ,
q

3.3 Monad operations

```
pmapM :: Monad m => (a->m b) -> d a -> m (d b)
fmapM :: Monad m => (a->m c) -> (b->m d) -> f a b -> m (f c d)
cataM :: Monad m => (FunctorOf d a b -> m b) -> (d a -> m b)
anaM  :: Monad m => (b -> m (FunctorOf d a b)) -> (b -> m (d a))
hyloM :: Monad m => (f a b -> m d) -> (c -> m (f a b)) -> c -> m d
paraM :: Monad m => (d a -> FunctorOf d a b -> m b) -> d a -> m b
```

pmapM pmap m

, x ,
T
w

Also known as: 11 **Known uses**: `unify`
Background: M
4 T ( ) `thread`
( ):

```
pmapM f  =  thread . pmap f
thread   =  pmapM id
```

```
propagate :: d (Maybe a) -> Maybe (d a)
cross     :: d  a  ->  d a
```

`propagate` `othing` `cross`
( ) `x` ,
T
w `x` T w
`thread` **Known uses**: `propagate`
fi `pzip` **Background**: `propagate`
`transpose` 19 , `propagate` `cross`
`thread` w

```
thread  :: Monad m => d (m a) -> m (d a)
fthread :: Monad m => f (m a) (m b) -> m (f a b)
```

`thread`
Also known as: $\mathtt{dist}_d$ 4 **Known uses**: `thread`
fi : `pmapM f = thread . pmap f` -
`fthread`
ff `thread` (w `d` =)
H k

`accumulate` `Monad m =>` `m a` `-> m` `a`

(w `m` = `Maybe`) `propagate` (w `m` =) `cross`

3.4 Flatten functions

```
flatten  :: d a ->  a
fflatten :: f a  a  ->  a
fl_par   :: f a b ->  a
fl_rec   :: f a b ->  b
```

`flatten x` `x`
T

f **Also known as**: extract$_{m,i}$ 11 , **listify** 6 **Known uses**: fl_rec
fi fi
fflatten fi flatten

```
flatten = cata fflatten
```

Background: I 6
mem.d ( ) **d** a **mem.d** x ≡
a ‘elem‘ (flatten x)

3.5 Miscellaneous

fi crush
pmap w P L w
, fi

```
psum    d Int -> Int                psum = crush (+) 0
prod    d Int -> Int                prod = crush (*) 1
conc    d  a  ->  a                 conc = crush (++)
pand    d Bool -> Bool              pand = crush (&&) True
por     d Bool -> Bool              por  = crush (  ) False

size       d a -> Int               size    = psum . pmap ( _->1)
flatten    d a ->  a                flatten = conc . pmap (   )
pall       (a->Bool) -> d a -> Bool pall p  = pand . pmap p
pany       (a->Bool) -> d a -> Bool pany p  = por  . pmap p
pelem      Eq a => a -> d a -> Bool pelem x = pany ( y->x==y)
```

4 Polytypic applications using PolyLib

T w w w
M P L

- w 7
- P , 12 , fi -
  “w - ” W H k HasWildcard
  w isWild t -> Bool x
  pmatch

  ```
  pmatch  HasWildcard t => t -> t -> Bool
  ```

  k w
- G w
  w w - fi
  T fi 9 : unify

```
unify    (Term t Subst s) => t -> t -> Maybe (s t)
```

k w ust fi
othing w T Term
unify

- G fi , w w : 13 w
  re rite

```
re rite    Term t =>  (t t)  -> t -> t
```

k w ( )
w

- ff 21 , w -

T -
x

5 Conclusions

W P L : P P T
w x w P L
k , w k w
w k -
,
P L P P,

http // .cs.chalmers.se/~patrikj/poly/polyp/

References

1 G B è, C B J , E M ML I *PLILP'96*, 1140 *LNCS* S -V , 1996

2 R S B I M B , , *Logic of Programming and Calculi of Discrete Design*, 36 *NATO ASI Series*, 5–42 S –V , 1987

3 R C k T k C Y w S R N 92/480/18, D C S , U C , 1992

4 M M kk M M - I 94-28, U Tw , J 1994

5 E G , R H , R J , J V *Design Patterns – Elements of Reusable Object-Oriented Software* -W , 1995

6 P H jk R B k W ? I *Category Theory and Computer Science*, LNCS 1290, 242–260, 1997

7 M k H T M ' , U U , 1996 IN /SRC-96-19

8 P J J J P P - x I *POPL'97*, 470–482 CM P , 1997

9 P J J J P fi *JFP*, 1998 I

10 P k J — - T , C U T , Sw , 1997 L http // .cs.chalmers.se/~patrikj/lic/

11 C B J , G B è, E M ML Ex 1 J P'98, 1998

12 J J P I *FPCA'95*, 238–248 CM P , 1995

13 J J P J P I *AFP'96*, 1129 *LNCS*, 68–114 S -V , 1996

14 G M H I J L S , , *Mathematics of Program Construction*, 335–347 S -V , 1989 LNCS 375

15 G M D *Science of Computer Programming*, 14:255–279, 1990

16 L M — w - I *Proceedings of the CWI Symposium on Mathematics and Computer Science*, 1 *CWI Monographs*, 289–334 N – H , 1986

17 L M P *Formal Aspects of Computing*, 4(5):413–425, 1992

18 L M C ! I *PLILP'96*, 1140 *LNCS*, 1–16 S V , 1996

19 z R *Analytical and Structural Polymorphism Expressed Using Patterns Over Types* P D , U M , 1992

20 T S *ACM TOPLAS*, 13(4):531–557, 1991

21 Må V G H k w M ' , Gö U , G , Sw , 1997 http // .cs.chalmers.se/~johanj/polytypism/genetic.ps