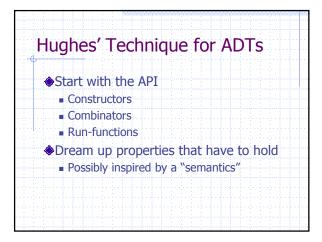


This Lecture A monadic parsing library An efficient choice operator Breadth-first rather than depth-first No special annotations Derived Used in GHC's Read class



```
Hughes' Technique (II)

First implementation

A big datatype
Each combinator becomes a constructor
Each run-function becomes an interpreter
Using properties, consecutively refine this
Identify usage patterns
Give them names
Simplify away the old ones
```

```
Parser API

type P s a

--parser combinators
symbol :: P s s
fail :: P s a
(+++) :: P s a -> P s a -> P s a

--monadic operators
return :: a -> P s a
(>>=) ::: P s a -> (a -> P s b) -> P s b
```

```
Implementation A

data P s a = Symbol  — wrong!

| Fail
| P s a :+++ P s a
| ∀b P s b :>>= (b -> P s a)
| Return a
```

```
Implementation A (as before)

data P s a where
Symbol :: P s s
Fail :: P s a
(:+++) :: P s a -> P s a -> P s a
(:>>=) :: P s a -> (a -> P s b) -> P s b
Return :: a -> P s a
```

```
Implementation A (III)

symbol = SymbolMap id
fail = Fail
(+++) = (:+++)
(>>=) = (:>>=)
return = Return
```

```
Implementation B

-- new constructor
SymbolBind k === symbol >>= k

-- new datatype
data P s a = SymbolBind (s -> P s a)
[ Fail
[ P s a :+++ P s a
[ Return a
```

```
Implementation C (II)

- unchanged operations
fail = Fail
symbol = SymbolBind return

-- new return
return x = ReturnPlus x Fail
```



```
Implementation C (V)

parse :: P s a -> [s] -> [(a, [s])]
parse (SymbolBind f) (c:s) = parse (f c) s
parse (SymbolBind f) [] = []
parse Fail s = []
parse (ReturnPlus x p) s = (x,s):parse p s
```

```
Implementation C, primed

type P' s a

symbol' :: P' s s
fail' :: P' s a
(+++') :: P' s a -> P' s a -> P' s a

return' :: a -> P' s a
(>>=') :: P' s a -> (a -> P' s b)

parse' :: P' s a -> [s] -> [(a,[s])]
```

```
Implementation D

-- context: "* >>=' k"

type Ctxt s a b = a -> P' s b

type P s a = ∀.Ctxt s a b -> P' s b

-- laws

p === \k . p' >>=' k

parse p === parse' p'
```

```
Implementation D (II)
-- parser combinators
symbol = \k. SymbolBind k
fail = \k. Fail
p +++ q = \k. p k +++' q k
-- monadic operators
return x = \k. k x
p >>= f = \k. p (\x. f x k)
-- run-function
parse p = parse' (p return')
```

```
Extension: Look Ahead

-- new operation
look:: P s [s]

-- semantics
[[look|] s = {(s,s)}

-- datatype
data P' s a = ...
| LookBind ([s] -> P' s a)
```

Extension: Look Ahead (II) -- add to +++' LookBind f +++' LookBind g = LookBind (\s.f s +++' g s) LookBind f +++' q = LookBind (\s.f s+++'q) p +++' LookBind f = LookBind (\s.p+++'f s) -- add to >>=' LookBind f >>=' k = LookBind (\s.f s>>='k) -- add to parse' parse' (LookBind f) s = parse' (f s) s

```
munch :: (s -> Bool) -> P s [s]
munch r = do s <- look; inspect s
where
inspect (c:s) | r c =
do symbol
s' <- inspect s
return (c:s')
inspect _ =
do return []</pre>
```

Usage of Look Ahead (II) longest :: Psa -> Psa try :: Psa -> Ps (Maybe a)

```
parse' :: P' s a -> [s] -> Either Pos a
parse' p s = track p s pos0

track (SymbolBind f) (c:s) pos =
   track (f c) s $! next pos c
   track (ReturnPlus x _) [] pos = Right x
   track (ReturnPlus _p) s pos = track p s pos
   track (LookBind f) s pos = track (f s) s pos
   track _ _ _ pos = Left pos
```

```
Discussion

Derivation vs. hacking
Efficiency
Parsec, GHC
Fudgets, SP a b
Continuation monads
Termination
```