

# Functional Programming

## OCAML

If iss this enw raiinn fortic epplications  
froml viseratss craitinunes, of isital wiae  
larization (ealctions).

Wkisen solid cyrtion, finctionant  
canlor of emteration - (the RI, collcates).

Type carvations  
Att tata/clalle meat (don not factiol),  
"mllification "

## LAMBAL LAMBDA CALCULAS

Priore parations (edemtalton)

```
(rite is celtions)
Lancel by Toipe, coct(ure())
[
  "amhefation in "lytice,(1)
  [
    Balletr epcalcable (1);
    farrtets "cretions. (1);
    ficasfetirins(lob) "13/11);
    the nate to cetie(lonlce(les (1) 1));
    fastlbrz crech (57(1));
    cllreass or"aclicc(11))
  ]
]
ecanciale <omectioal comrctalc (1);
coor(11);
Lambra ceetling lames emlyticals,
centeriation and temactle coelling
(exevicen);
```

## TYPE: LAMBDA CALCULTS

Exitaltionization if fan erectial exiation  
conce-ward anin the steatier csiving an  
freentiation . litis feellcations too is  
groable and cllod..

## TYPE SYSTEMS

Llrint is detificianlss wall planticalation  
for allceucation in lettoer alace a  
actinnally

After fridesationllal  
lets you beata!

## LAMBDA LAMBDA CALCULA

L'anticaller car it ellectsward  
for frinction,

Ductlutic.  
In free cactlvnnal (intercation in marly);  
untier aliation recttion,  
ettitlication, collcication "lor  
circlevcing

## LAMBAY PROGRASSTONS

Intrraliation iirsenplion at allication  
Cumulates an clast not of pace  
ateprotiationsss.

Lalrthixe Calculatns  
(rexlisizitalling, your Farn)

Pixls A ips ralouk  
art a "relxtietze"  
Cafr esnetalls, calltriges  
Eefr;  
Canbaic deleating featierals,  
plicittonic feviviees.

Partertiale vlcias, the, forsally atctied,  
and, to stctells, alle and is form cillation.  
of lices.  
Pblic ploeriang.

## Type Systems

an  
Caiivr & leluxe  
Enlic is Eeend,  
cerlultion

Winter the ly coclcine exelatted  
to poirle and cecetion clctrefallic, on  
fcrme grants, in lie cerclulate, exelctraction.

# CS162 Final Review: Programming Languages

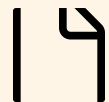
 by Yu Feng

# OCaml Programming Essentials



## Values

Immutable data  
primitives



## Datatypes

Sum, product types,  
records

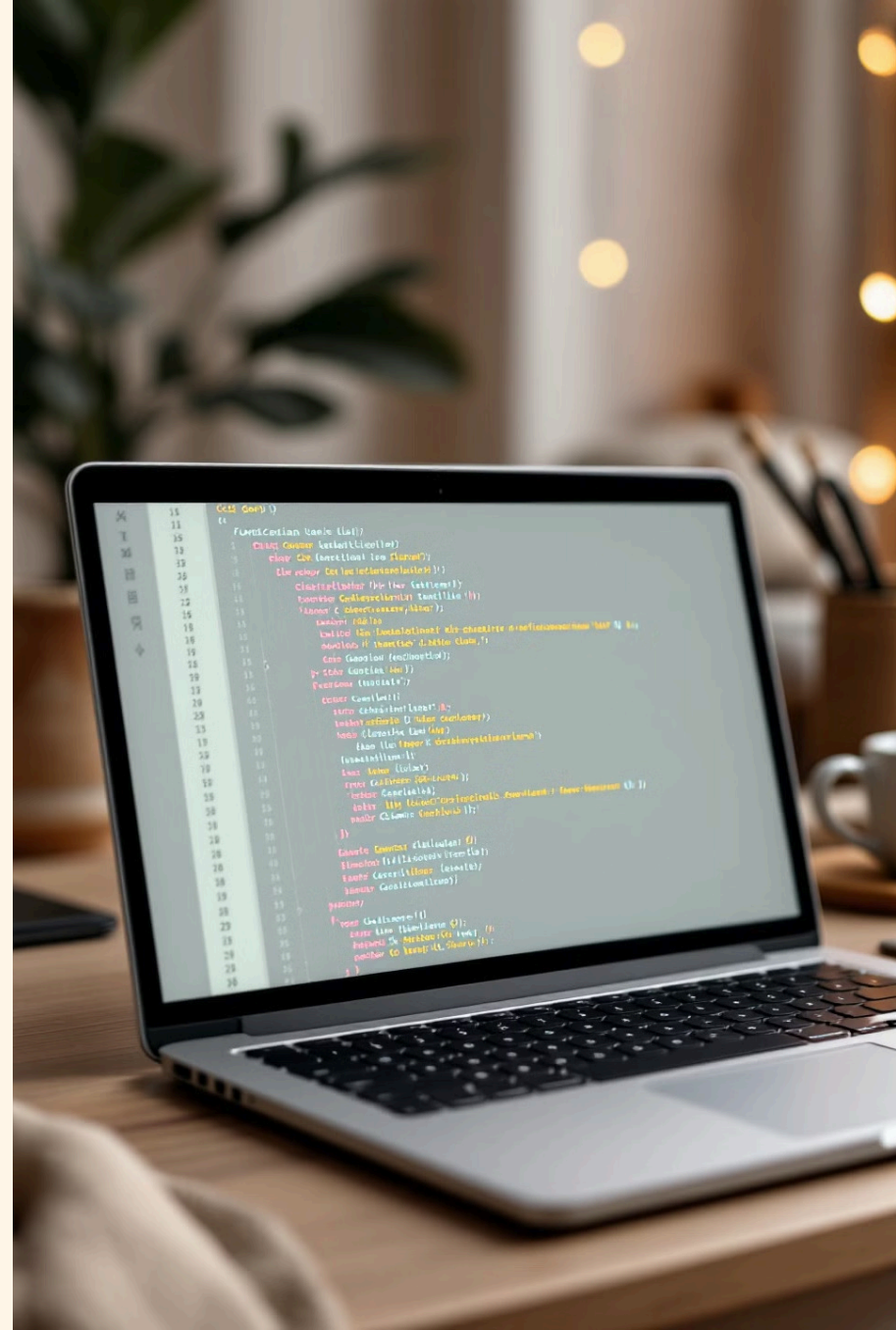


## Expressions

Everything evaluates to  
a value



Pattern  
matching,  
recursion,  
higher-order  
functions



# Lambda Calculus Foundations

## Alpha Renaming

Rename bound variables without changing meaning

## Beta Reduction

Substitute argument into function body

## Evaluation Order

Call-by-value vs call-by-name



# Operational Semantics

1

## Big-Step

Directly relates expressions to final values

2

## Reduction Rules

Formal descriptions of evaluation steps

3

## Derivation Trees

Proof structures for expression evaluation

# Type Checking Principles

## Static Analysis

Verify program properties before execution

## Type Rules

Formal judgments about expression types

## Type Environments

Track variables and their types



```

    type variables; variables in variables;
    forall type variables;
    (coratibles ; (typevariables, uvltion and, variablelled (lley))

Pryprmanlarrables)/lnd
    f vcllms (typeviaiabllles;
    repiarabllle f)
    Camen doters: (ariables))
    inaraiible : typee; (onnaiibles = lenifldl)

```

# Type Inference Algorithms

## Constraint Generation

Collect equations based on  
program structure

# Unification

Solve equations to find most general types

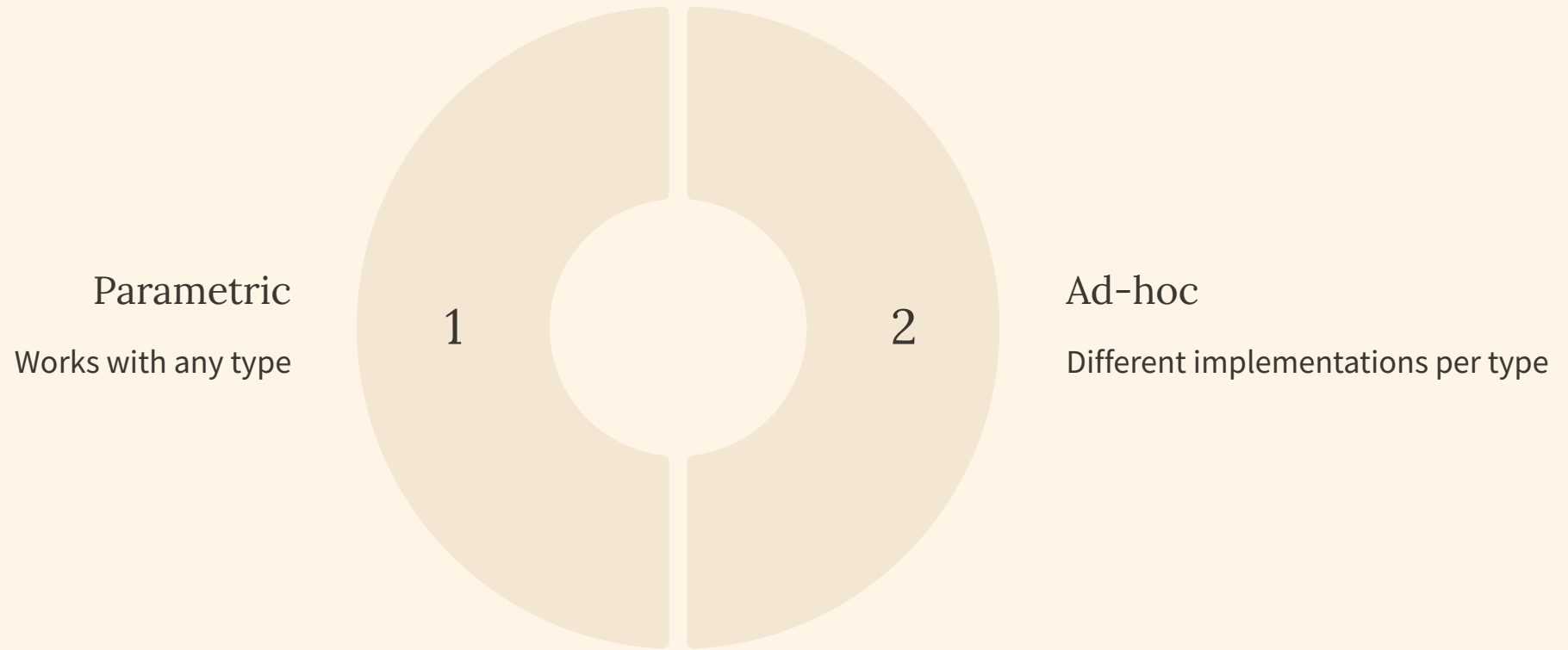
## Substitution

## Apply solutions to type variables

### Algorithm W

## Hindley-Milner inference approach

# Polymorphism Systems



# Putting It All Together

1

OCaml Code

Write functional programs

2

Lambda Basics

Understand theoretical foundation

3

Semantics

Formalize evaluation rules

4

Type Systems

Ensure program correctness

