UMass Boston Computer Science

CS450 High Level Languages (section 2)

User-defined “Lambda” Functions in CS450 Lang

Monday, November 25, 2024

Logistics

- HW 11 in
  - ~~due: Mon 11/25 12pm noon EST~~

- HW 12 out
  - due: **Wed** 12/4 12pm noon EST

Function Application in CS450 Lang

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Variable Expr] Expr)
;; - ‘(Expr . List<Expr>)
```

Function call case (must be last, why?)

“cons”

be careful when parsing this in HW!

What functions can be called?

Function Application in CS450 Lang

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Variable Expr] Expr)
;; - ‘(Expr . List<Expr>)
```

```
;; An Environment (Env) is one of:
;; - empty
;; - (cons (list Var Result) Env)
```

```
;; A Result is a:
;; - Number
;; - ErrorResult
;; - (Racket) Function
```

What functions can be called?

```
(+ 1 2 3)
```

```
(define INIT-ENV
  `((+ ,450+)
    (- ,450-)))
```

(Racket) functions, added to initial environment

These should now have **“variable arity”** (like **Racket +/-**)

Interlude: Variable-arity functions in Racket

Programmer should not be constructing a list

```
;; 450+: List<Result> -> Result ???
```

```
;; 450+: Result … -> Result
```

```
(define/contract (450+ . args)
  (-> Result? ... Result? )
    …    )
```

Inside the function, **args** is a **list of arguments**

These should now have **“variable arity”** (like **Racket +/-**)

(compare with JS “variadic” args)

```
function sum(...theArgs) {
  let total = 0;
  for (const arg of theArgs) {
    total += arg;
  }
  return total;
}
```

Function Application in CS450 Lang

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Variable Expr] Expr)
;; - ‘(Expr Expr ...)
```

“zero or more of the preceding”

parse

```
;; An AST is one of:
;; …
;; - (call AST List<AST>)
;; …
(struct call [fn args])
```

run

```
;; A Result one of:
;; - …
```

“Running” Function Calls

```
;; run: AST -> Result
(define (run p)


  (define (run/e p env)
    (match p
         …
     [(call fn args) (apply
                       (run/e fn env)
                       (map (curryr run/e env) args))]

         …
     ))
 (run/e p INIT-ENV))
```

```
;; An AST is one of:
;; …
;; - (call AST List<AST>)
;; …
(struct call [fn args])
```

“Running” Function Calls

```
;; run: AST -> Result
(define (run p)


  (define (run/e p env)
    (match p
        …
     [(call fn args) (apply
                        (run/e fn env)
                        (map (curryr run/e env) args))]

        …
     ))
 (run/e p INIT-ENV))
```

```
;; An AST is one of:
;; …
;; - (call AST List<AST>)
;; …
(struct call [fn args])
```

TEMPLATE: extract pieces of compound data

“Running” Function Calls

```
;; run: AST -> Result
(define (run p)

  (define (run/e p env)
    (match p
       …
     [(call fn args) (apply
                          (run/e fn env)
                          (map (curry??? run/e env) args))]
       …
     ))
 (run/e p INIT-ENV))
```

```
;; An AST is one of:
;; …
;; - (call AST List<AST>)
;; …
(struct call [fn args])
```

TEMPLATE: recursive calls

List-processing function

“Running” Function Calls

How do we actually run the function?

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
```

```
(define (run p)



  (define (run/e p env)
    (match p
       ...
     [(call fn args) (apply
                      (run/e fn env)
                      (map (curryr run/e env) args))]
       ...
     ))
 (run/e p INIT-ENV))
```

Runs a Racket function

???

function

List of args

(this only “works” for now)

Function Application in CS450 Lang

What functions can be called?

```
(+ 1 2)
```

```
(??? 1 2)
```

1. (Racket) functions in initial environment

2. user-defined (“lambda”) functions?

“Lambdas” in CS450 Lang

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Var Expr] Expr)
;; - ‘(Expr Expr ...)
```

“Lambdas” in CS450 Lang

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Var Expr] Expr)
;; - ‘(fn (Var ...) Expr)
;; - ‘(Expr Expr ...)
```

CS450 Lang “Lambda” examples

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Var Expr] Expr)
;; - ‘(fn (Var ...) Expr)
;; - ‘(Expr Expr ...)
```

CS450LANG

```
(fn (x y) (+ x y))
```

Equivalent to ...

RACKET

```
(lambda (x y) (+ x y))
```

```
(fn (x) (fn (y) (+ x y)) ; “curried”
```

```
((fn (x y) (+ x y))
 10 20 ) ; fn applied
```

CS450 Lang “Lambda” full examples

```
(check-equal?
  (eval450
   ‘(bind [x 10]
      ( (fn (y) (+ x y)) 20 )))
   30 ) ; with bind
```

Expression that evaluates to a function result

argument

```
(check-equal?
  (eval450
   ‘( (bind [x 10]
        (fn (y) (+ x y)))
      20 ))
   30 ) ; with bind (fn only)
```

```
(check-equal?
  (eval450
   ‘( (fn (x y) (+ x y))
        10 20 ) )
   ? )
```

- Repo: **cs450f24/in-class-25**
- File: **hw12-examples-**<Last>**-**<First>**.rkt**

In-class Coding 11/25: `fn` examples

```
(check-equal?
  (eval450
   ‘(bind [x 10]
      ( (fn (y) (+ x y)) 20 )))
   30 ) ; with bind
```

```
(check-equal?
  (eval450
   ‘( (bind [x 10]
        (fn (y) (+ x y)))
      20 ))
    30 ) ; with bind (fn only)
```

Expression that evaluates to a function result

argument

```
(check-equal?
  (eval450
   ‘( (fn (x y) (+ x y))
       10 20 ) )
  30 )
```

Come up with some of **your own!** (i.e., not my examples)

5 meaningful examples only! with **eval450**

```
;; An Expr is one of:
;; - Atom
;; - Variable
;; - ‘(bind [Var Expr] Expr)
;; - ‘(fn (Var ...) Expr)
;; - ‘(Expr Expr ...)
```

CS450 Lang “Lambda” AST node

```
;; A 450LangExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - ‘(bind Var Expr Expr)
;; - ‘(fn (Var ...) Expr)
;; - ‘(Expr Expr ...)
```

parse

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

Why can’t we use a Racket `lambda`?

A Racket `lambda` is a “Result”, e.g., you can’t “get” the parameters or the body code (it’s not “transparent”)

“Running” Functions?

```
;; run: AST -> Result
(define (run p)

  (define (run/e p env)
    (match p
        …
     [(fn-ast params body) ?? params ?? (run/e body env) ??]
        …
     ))
 (run/e p INIT-ENV))
```

TEMPLATE

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

“Running” Functions?

```
;; run: AST -> Result
```

```
(define (run p)


  (define (run/e p env)
    (match p
        …
     [(fn-ast params body) ?? params ?? (run/e body env) ??]
    ))
 (run/e p
```

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function ???
```

What should be the “Result” of running a function?

Can we “convert” a 450lang “fn” AST into a Racket function???

We can’t!! (it’s not “transparent”) (this is what makes FFIs and mixed lang progs complicated) So we need some other representation

"Running" Functions?

Can we "convert" this into a Racket function?

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

WAIT! Are **fn-result** and **fn-ast** the same?

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
;; - (fn-result List<Symbol> AST ??)
(struct fn-result [params body])
```

We can't!! need some other representation

“Running” Functions? Full example

```
(bind [x 10]
  (fn (y) (+ x y)))
```

parse

```
(bind 'x (num 10)
  (fn-ast '(y)
    (call (var '+)
          (list (var 'x) (var 'y)))
```

run

```
(fn-result '(y)
  (call (var '+)
        (list (var 'x) (var 'y))
```

Where is the x???

`fn-result` and `fn-ast` cannot be the same!!

(how can we **“remember”** the **x**)

“Running” Functions?

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

WAIT! Are `fn-result` and `fn-ast` the same?

```
;; A Result is one of:
;; - Number
;; - ErrorREsult
;; - (Racket) Function
;; - (fn-result List<Symbol> AST ???)
(struct fn-result [params body])
```

"Running" Functions?

Takeaway quiz:
Q: What is the difference between **`fn-ast`** and **`fn-result`**?

A: **`fn-ast`** is AST data, represents code that a programmer writes;
`fn-result` is `Result` data, represents result of running the program
(importantly contains **environment** for variables that are not fn parameters)

A Function `Result` needs an extra environment
(for the non-argument variables used in the body!)

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

"Running" Functions?

```
;; run: AST -> Result
(define (run p)


  (define (run/e p env)
    (match p
         …
     [(fn-ast params body) ?? params ?? (run/e body env) ??]
```

What should be the "Result" of running a function?

```
    ))
 (run/e p
```

Can we "convert" a 450lang "fn" AST into a Racket function???

We can't!! need some other representation

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function ???
```

“Running” Functions?

```
;; run: AST -> Result
(define (run p)


  (define (run/e p env)
    (match p
        …
     [(fn-ast params body) ?? params ?? (run/e body env) ??]

     ))
 (run/e p INIT-ENV))
```

```
;; An AST is one of:
;; …
;; - (fn-ast List<Symbol> AST)
;; …
(struct fn-ast [params body])
```

What should be the “Result” of running a function?

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

Result of "Running" a Function

```
;; run: AST -> Result
(define (run p)


  (define (run/e p env)
    (match p
        ...
     [(fn-ast params body) (fn-result params body env)]
        ...
     ))
 (run/e p INIT-ENV))
```

body won't get "run" until the function is called

Save the current **env**

Previously

"Running" Function Calls: Revisited

How do we actually run the function?

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
```

???

Runs a Racket function

```
(define (run p)


  (define (run/e p env)
    (match p
        ...
     [(call fn args) (apply
                       (run/e fn env)
                       (map (curryr run/e env) args))]
        ...
     ))
 (run/e p INIT-ENV))
```

(this only "works" for now)

“Running” Function Calls: Revisited

How do we actually run the function?

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
(define (run p)


  (define (run/e p env)
    (match p
         …
      [(call fn args) (450apply
                        (run/e fn env)
                        (map (curryr run/e env) args))]
         …
      ))
  (run/e p INIT-ENV))
```

apply doesn’t work for fn-result!!
must manually implement “function call”

(this doesn’t “work” anymore!)

CS450 Lang "Apply"

Can we refactor data def to make this cleaner?

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : [Racket fn or fn-result] List<Result> -> Result
(define (450apply fn args)
 …
)
```

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - FnResult
```

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

CS450 Lang "Apply"

TEMPLATE?

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
  …
)
```

CS450 Lang "Apply"

TEMPLATE

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)          …          ] ;; racket function
  [(fn-result params body env)          ;; user-defined function
        …   params         …        body         …          env]))
```

CS450 Lang “Apply”

TEMPLATE: mutually referential data and template calls!

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)          …          ] ;; racket function
  [(fn-result params body env)        ;; user-defined function
      …      params    …        (ast-fn body … ) … (env-fn env … ) … ]))
```

```
env-add : Env Var Result -> Env
```

CS450 Lang "Apply"

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)        …        ] ;; racket function
  [(fn-result params body env)      ;; user-defined function
     …    (ast-fn body … ) … (env-add env ?? args params ?? ) … ]))
```

These are lists

```
env-add : Env Var Result -> Env
```

CS450 Lang "Apply"

(so this function should be inside `run`)

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)          …          ] ;; racket function
  [(fn-result params body env)        ;; user-defined function
      …     (ast-fn body … ) … (foldl env-add env params args) … ]))
```

`run/e : AST Env -> Result`

CS450 Lang "Apply"

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?)            ???           ] ;; racket function
                             ...
  [(fn-result params body env)     ;; user-defined function
   (run/e body (foldl env-add env params args))]))
```

`run/e : AST Env -> Result`

CS450 Lang “Apply”

```
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?) (apply fn args)] ;; racket function
  [(fn-result params body env)     ;; user-defined function
   (run/e body (foldl env-add env params args))]))
```

Runs a Racket function

WAIT! What if the the number of params and args don’t match!

CS450 Lang "Apply"

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?) (apply fn args)] ;; racket function
  [(fn-result params body env)     ;; user-defined function
   (if (= (length params) (length args))
       (run/e body (foldl env-add env params args))
       …   ]))
```

CS450 Lang “Apply”: arity error

```
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
 (match fn
  [(? procedure?) (apply fn args)] ;; racket function
  [(fn-result params body env)     ;; user-defined function
   (if (= (length params) (length args))
       (run/e body (foldl env-add env params args))
       ARITY-ERROR)]))
```

```
;; An ErrorResult is one of:
;; - UNDEFINED-ERROR
;; - NOT-FN-ERROR
;; - ARITY-ERROR
```

```
;; A Result is one of:
;; - Number
;; - ErrorResult
;; - FnResult
```

- Repo: **cs450f24/in-class-25**
- File: **hw12-examples-**<Last>**-**<First>**.rkt**

In-class Coding 11/25: `fn` examples

```
(check-equal?
  (eval450
   '(bind [x 10]
      ( (fn (y) (+ x y)) 20 )))
   30 ) ; with bind
```

```
(check-equal?
  (eval450
   '( (bind [x 10]
        (fn (y) (+ x y)))
      20 ))
    30 ) ; with bind (fn only)
```

Expression that evaluates to a function result

argument

```
;; An Expr is one of:
;; - Atom
;; - Variable
;; - '(bind [Var Expr] Expr)
;; - '(fn (Var ...) Expr)
;; - '(Expr Expr ...)
```

```
(check-equal?
  (eval450
   '( (fn (x y) (+ x y))
        10 20 ) )
   30 )
```

Come up with some of **your own!** (i.e., not my examples)

5 meaningful examples only! with **eval450**