Interpreting Lambda Functions

Wednesday, November 29, 2023
Logistics

- HW 8 out
  - due: Sun 12/3 11:59 pm EST
  - hw-start repo
    - has tests-from-lecture23.rkt file
Interlude: What is a “binding”? 

In programming, a **binding** is an association of an **identifier** with a value. Not all bindings are **variables** — for example, function **parameters** and the binding created by the **catch** block are not "variables" in the strict sense. In addition, some bindings are implicitly created by the language — for example, **this** and **new.target** in JavaScript.

A binding is **mutable** if it can be re-assigned, and **immutable** otherwise; this does not mean that the value it holds is immutable.

A binding is often associated with a **scope**. Some languages allow re-creating bindings (also called redeclaring) within the same scope, while others don’t; in JavaScript, whether bindings can be redeclared depends on the construct used to create the binding.

"bind" in "CS450js" Lang: New Syntax!

;; A Variable (Var) is a Symbol

;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list 'bind [Variable Expr] Expr)

Reference a variable binding
Create new variable binding (now with extra brackets!)
new binding is in-scope here

(bind [x 10] (+ x 1))
Equivalent to ...
(let ([x 10]) (+ x 1))
Bind scoping examples

;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list ‘bind [Variable Expr] Expr)

bind obeys “lexical” or “static” scoping

Generally accepted to be “best choice” for programming language design
(bc it’s determined only by program syntax)

(check-equal? (eval450 ‘(bind [x 10] x))
10 ); no shadow

(check-equal? (eval450 ‘(bind [x 10]
  (bind [x 20] x))
20 ); shadow

(check-equal? (eval450 ‘(bind [x 10]
  (+ (bind [x 20] x)
  x))
30 )

(check-equal? (eval450 ‘(bind [x 10]
  (bind [x (+ x 20)] x)))
30 )
Running `bind` programs

```plaintext
;; A 450jsExpr (Expr) is one of: 
;; - Atom 
;; - Variable 
;; - (list `bind [Variable Expr] Expr)
```

```
parse

;; A 450jsAST (AST) is one of:
;; ...
;; - (bind Symbol AST AST)
;; ...
(struct bind [var expr body])
```

```
run

;; A 450jsResult (Result) is a:
;; - ...
```
Running **bind**

```scheme
;; run: AST -> Result
(define (run p)
  (define (run/e p env)
    (match p
     ...
     ...
     [(bind x e body) ?? x ?? e ?? body ])
    (run/e p ??? )))
```

;; A 450jsAST (AST) is one of:
;; ... ;; - (bind Symbol AST AST)
;; ... (struct bind [var expr body])

```
```
Running `bind`

``` Scheme
;; run: AST -> Result
(define (run p)
  (define (run/e p env)
    (match p
      ...
      [(bind x e body) x (run/e e) (run/e body)]
      ...
      (run/e p)))
```
Running **bind**, using environment

```scheme
;; run: AST -> Result
(define (run p)
  ;; An Environment (Env) is one of:
  ;; - empty
  ;; - (cons (list Var Result) Env)

  ;; accumulator env : Environment
  (define (run/e p env)
    (match p
      ...
      [(bind x e body) ?? x ?? (run/e e ??) ?? (run/e body ??) ]
      ...
    )
    (run/e p ??? ))
)
```
Running **bind**, using environment

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

;;; accumulator env : Environment
(define (run/e p env)
  (match p
    ...
    [(bind x e body) ?? x ?? (run/e e env) ?? (run/e body ??) ]
    ... ))
  (run/e p ??? ))
Running \textbf{bind}, using environment

\texttt{;; run: AST -> Result}

\begin{verbatim}
(define (run p)

;; accumulator \texttt{env} : Environment
(define (run/e p env)
  (match p
    ...
    [(bind x e body)
      (define new-env (env-add env x (run/e e env))
        (run/e body ???
          ... )))
  (run/e p ??? ))
)
\end{verbatim}
Running \textbf{bind}, using environment

\begin{verbatim}
;; run: AST -> Result
(define (run p)

;; accumulator env : Environment
(define (run/e p env)
  (match p
    ...
    [(bind x e body)
      (define new-env (env-add env x (run/e e env))
       (run/e body new-env))
    ...
    ])
  (run/e p ??? ))

3. run body with new env
   (x in-scope)
\end{verbatim}
Function Application in CS450js

A 450jsExpr (Expr) is one of:

- Atom
- Variable
- (list ‘bind [Variable Expr] Expr)
- (cons Expr List<Expr>)

Function call case (must be last, why?)

be careful when parsing this (HW 8!)

What functions can be called?
Function Application in CS450js

;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Variable
;; - (list 'bind [Variable Expr] Expr)
;; - (cons Expr List<Expr>)

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

;; A 450jsResult (Result) is a:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function

What functions can be called?

(+ 1 2)

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

(Racket) functions, added to initial environment
Function Application in CS450js

A 450jsExpr (Expr) is one of:

- Atom
- Variable
- (list ‘bind [Variable Expr] Expr)
- (cons Expr List<Expr>)

A 450jsAST (AST) is one of:

- ... (call AST List<AST>)
- ... (struct call [fn args])

A 450jsResult (Result) is a:

- ...
“Running” Function Calls

(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))
;; 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))

;; A 450jsAST (AST) is one of:
;; ...
;; - (call AST List<AST>)
;; ...
;; (struct call [fn args])

 TEMPLATE: recursive calls

 List-processing function
“Running” Function Calls

(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))

How do we actually run the function?

;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function

Runs a Racket function

(this only “works” for now)
A 450jsExpr (Expr) is one of:
- Atom
- Var
- (list ‘bind [Var Expr] Expr)
- (cons Expr List<Expr>)

What functions can be called?

1. (Racket) functions added to initial environment
2. user-defined ("lambda") functions?
“Lambdas” in CS450js

;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Var
;; - (list 'bind [Var Expr] Expr)
;; - (list 'fn List<Var> Expr)
;; - (cons Expr List<Expr>)
CS450js “Lambda” examples

;; A 450jsExpr (Expr) is one of:
;; - Atom
;; - Var
;; - (list ‘bind [Var Expr] Expr)
;; - (list ‘fn List<Var> Expr)
;; - (cons Expr List<Expr>)

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

Equivalent to ...

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

(fn (x) (fn (y) (+ x y))) ; “curried”

((fn (x y) (+ x y))
  10 20) ; fn applied
CS450js “Lambda” full examples

(expression that evaluates to a function result)

(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)

(check-equal? (eval450 ‘((fn (x y) (+ x y)) 10 20 ) ) 3) )
In-class Coding 11/29: fn scope examples

Expression that evaluates to a function result

Come up with some of your own!
CS450js “Lambda” AST node

;;;; A 450jsExpr (Expr) is one of:
;;;; - Atom
;;;; - Variable
;;;; - (list ‘bind Var Expr Expr)
;;;; - (list ‘fn List<Var> Expr)
;;;; - (cons Expr List<Expr>)

;;;; A 450jsAST (AST) is one of:
;;;; ... 
;;;; - (fn-ast List<Symbol> AST)
;;;; ... 
;;;; (struct fn-ast [params body])

parse
“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))

;; A 450jsAST (AST) is one of:
;; ...
;; - (fn-ast List<Symbol> AST)
;; ...
;; [struct fn-ast [params body]]
"Running" Functions?

```scheme
;;; 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?

  Can we "convert" a 450js program AST into a Racket function???

  We can't!! So we need some other representation

  ;; A 450jsAST (AST) is one of:
  ;; ...
  ;; - (fn-ast List<Symbol> AST)
  ;; ...
  (struct fn-ast [params body])

  ;; A 450jsResult is one of:
  ;; - Number
  ;; - UNDEFINED-ERROR
  ;; - (Racket) Function
```
“Running” Functions?

Can we “convert” this into a Racket function?

\[
\text{A 450jsAST (AST) is one of:} \\
\text{...} \\
\text{(fn-ast List<Symbol> AST) } \\
\text{...} \\
\text{(struct fn-ast [params body])}
\]

WAIT! Are \text{fn-result} and \text{fn-ast} the same?

\[
\text{A 450jsResult is one of:} \\
\text{... } \\
\text{- Number} \\
\text{- UNDEFINED-ERROR} \\
\text{- (Racket) Function} \\
\text{- (fn-result List<Symbol> AST ??)} \\
\text{(struct fn-result [params body])}
\]

\text{We can't!!} So we 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?

### 450jsResult

- Number
- UNDEFINED
- ERROR
- (Racket) Function
- (fn-result List<Symbol> AST)

### 450jsAST

- (fn-ast List<Symbol> AST)
- (struct fn-ast [params body])

**WAIT! Are fn-result and fn-ast the same?**
“Running” Functions?

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

;;; A 450jsResult is one of:
;;; - Number
;;; - UNDEFINED-ERROR
;;; - (Racket) Function
;;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
“Running” Functions?

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

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

We can't!! So we need some other representation
```

;; A 450jsAST (AST) is one of:
;; ...
;; - (fn-ast List<Symbol> AST)
;; ...
;; (struct fn-ast [params body])

;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function ??
"Running" Functions?

```
;; run: AST -> Result
(define (run p)
  ...
  [(fn-ast params body) ?? params ?? (run/e body env) ??])
)
```

```
;; A 450jsAST (AST) is one of:
;; ...
;; - (fn-ast List<Symbol> AST)
;; ...
(struct fn-ast [params body])
```

```
;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```

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

;; 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)))

Remember the current env

body won’t get “run” until the function is called
“Running” Function Calls: Revisited

(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))

---

How do we actually run the function?

;;; A 450jsResult is one of:
;;; - Number
;;; - UNDEFINED-ERROR
;;; - (Racket) Function

---

Runs a Racket function

(this only “works” for now)
“Running” Function Calls: Revisited

(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))

;; A 450jsResult is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
;; (struct fn-result [params body env])

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

(this doesn’t “work” anymore!)
Can we refactor data def to make this cleaner?

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

A `450jsResult` (Result) is one of:
- Number
- UNDEFINED-ERROR
- FnResult

A `FnResult` is one of:
- (Racket) Function
- (fn-result List<Symbol> AST Env)
  (struct fn-result [params body env])
```scheme
;; 450apply : FnResult List<Result> -> Result
(define (450apply fn args)
  ...
)
```

```scheme
;; A FnResult is one of;
;; - (Racket) Function
;; - (fn-result List<Symbol> AST Env)
(struct fn-result [params body env])
```
CS450JS Lang “Apply”

;; 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])))
CS450JS Lang “Apply”

```scheme
;; 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 ...) ... ]))
```

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

TEMPLATE: mutually referential data and template calls!
450apply : FnResult List<Result> -> Result
(define (450apply fn args)
  (match fn
    [(? procedure?) ... ] ;; racket function
    [([fn-result params body env)] ... (ast-fn body ... ) ... (env-add env ?? args params ?? ) ... ]))
CS450JS 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 ...) ... (foldl env-add env params args) ... ]))
```

(see this function should be inside run)
CS450JS 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))]))
CS450JS Lang “Apply”

```scheme
;; 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))]))

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

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

Runs a Racket function
CS450JS 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))
        ... ]))
CS450JS Lang “Apply”: arity error

```racket
;; 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))])
```

;; A 450jsResult (Result) is one of:
;; - Number
;; - UNDEFINED-ERROR
;; - ARITY-ERROR
;; - FnResult
No More Quizzes!

but push your in-class work to:
Repo: cs450f23/lecture24-inclass