Principles of Programming Languages

CS 314

Recitation 10

Topics Today

Fixed-point Combinator

Project 2

- Review of Scheme
- map, assoc, let, let*, letrec
- FIRST & FOLLOW algorithms
- Advice/Techniques

Topics Today

Fixed-point Combinator

Project 2

- Review of Scheme
- map, assoc, let, let*, letrec
- FIRST & FOLLOW sets algorithms
- Advice/Techniques

Fixed-point Combinator

Is there a way to "compute" the fixed point of any function F?

$$x = F(x)$$

YES. x = YF, and Y is called the fixed-point combinator.

$$Y \equiv \lambda f.((\lambda x.f(x\ x))\ (\lambda x.f(x\ x)))$$
$$YF = ((\lambda f.((\lambda x.f(\ x\ x))\ (\lambda x.f(\ x\ x\ ))))\ F)$$
$$YF = (\lambda x.F(x\ x))\ (\lambda x.F(x\ x))$$
$$YF = F((\lambda x.F(x\ x))\ (\lambda x.F(x\ x)))$$
$$YF = F(YF)$$

Fixed-point Combinator

Suppose I wanted to find out what happens to F when I unroll the recursion

F ≡ λf. (λmn. if (= n 1) then m else (f (* m n) (- n 1)))

((YF) 1 3) = ?

= (((λf.((λx.f(x x)) (λx.f(x x)))) F) 1 3)

= ((F (YF)) 1 3) unroll the recursion

= ((λmn. if (= n 1) then m else (YF (* m n) (- n 1))) 1 3) substitute YF for f

= if (= 3 1) then 1 else (YF (* 1 3) (- 3 1))

= (YF 3 2) = ((F (YF)) 3 2)

= ((λmn. if (= n 1) then m else (YF (* m n) (- n 1))) 3 2)

= if (= 2 1) then 3 else (YF (* 3 2) (- 2 1))

= (YF 6 1) = ((F (YF)) 6 1)

= ((λmn. if (= n 1) then m else (YF (* m n) (- n 1))) 6 1)

= if (= 1 1) then 6 else (YF (* 6 1) (- 1 1))

= 6

Topics Today

Fixed-point Combinator

Project 2

- Review of Scheme
- map, assoc, let, let*, letrec
- FIRST & FOLLOW sets algorithms
- Advice/Techniques

Project 2: Review of Scheme

car: Returns the first element of the list.

(car ‘(1 2 3 4)) = 1

cdr: Returns a list of the rest of the list (excluding the first element)

(cdr ‘(1 2 3 4)) = ‘(2 3 4)

What does this print? (cadr ‘(1 (2 3) 4 5))

Project 2: Review of Scheme

car: Returns the first element of the list.

(car ‘(1 2 3 4)) = 1

cdr: Returns a list of the rest of the list (excluding the first element)

(cdr ‘(1 2 3 4)) = ‘(2 3 4)

What does this print? (cadr ‘(1 (2 3) 4 5))

‘((2 3))

cons: Constructs a list

(cons ‘a ‘(1 2 3 4)) = ‘(a 1 2 3 4)

append: Takes two lists and appends the second to the first.

(append ‘(a b c) ‘(1 2 3 4)) = ‘(a b c 1 2 3 4)

list: Takes the arguments and puts it all into a list

(list ‘A ‘::= ‘delta) = ‘(A ::= delta)

(list ‘A ‘(::= delta)) = ‘(A (::= delta))

Topics Today

Fixed-point Combinator

Project 2

- Review of Scheme
- map, assoc, let, let*, letrec
- FIRST & FOLLOW sets algorithms
- Advice/Techniques

Project 2: map, assoc, let, let*, letrec

map: Takes a function and a list, and applies each element of the list with the function. Returns the list of newly applied elements.

(map even? ‘(1 2 3 4)) = ‘(#f #t #f #t)

What does this print?

(map (lambda (x) (+ x 3)) ‘(1 2 3 4))

Project 2: map, assoc, let, let*, letrec

map: Takes a function and a list, and applies each element of the list with the function. Returns the list of newly applied elements.

(map even? ‘(1 2 3 4)) = ‘(#f #t #f #t)

What does this print?

(map (lambda (x) (+ x 3)) ‘(1 2 3 4))

= ‘(4 5 6 7)

Project 2: map, assoc, let, let*, letrec

Associative lists: A list of pairs (though the pairs are themselves list) consisting of the first element being a symbol, and the second element the corresponding list.

Example:

```
‘(
   (a (1 2 3 4))
   (b (5 6 7))
   (c (45 10))
 )
```

What is the list associated with the symbol ‘a?

Project 2: map, assoc, let, let*, letrec

Associative lists: A list of pairs (though the pairs are themselves list) consisting of the first element being a symbol, and the second element the corresponding list.

Example:

```
‘(
   (a (1 2 3 4))
   (b (5 6 7))
   (c (45 10))
 )
```

What is the list associated with the symbol ‘a?

‘(1 2 3 4)

Project 2: map, assoc, let, let*, letrec

When are associated lists useful?

Remember that each non-terminal NT has an associated FIRST set.

So, for nonterminals a, b, c, the structure of the list of FIRST sets can look like the following:

```
'(
   (a FIRST(a))
   (b FIRST(b))
   (c FIRST(c))
 )
```

Similarly for FOLLOW sets.

How can we access FIRST(a) quickly and easily?

Project 2: map, assoc, let, let*, letrec

assoc: Takes as inputs an associative list and a symbol, and returns a list of the symbol itself and the corresponding list of the symbol.

So, for nonterminals a, b, c, suppose aList, an associative list, has the following form:

```
aList = ‘(
          (a FIRST(a))
          (b FIRST(b))
          (c FIRST(c))
         )
```

Then,

(assoc ‘a aList) = ‘(a FIRST(a))

How would you use assoc, car and cdr to get FIRST(a)?

Project 2: map, assoc, let, let*, letrec

```
aList = '(
          (a FIRST(a))
          (b FIRST(b))
          (c FIRST(c))
         )
```

Then,

(assoc ‘a aList) = ‘(a FIRST(a))

How would you use assoc, car and cdr to get FIRST(a)?

(cadr (assoc ‘a aList))

Project 2: map, assoc, let, let*, letrec

- let:
  - binds variables to values (no specific order), and evaluates body using bindings
  - new bindings are not effective during evaluation for the next binding
- let*:
  - binds variables to values in textual order of write-up
  - new binding is effective for next binding.
- letrec:
  - bindings of variables to values in no specific order
  - independent evaluations of all bindings to values have to be possible
  - mainly used for recursive function definitions

Example of let:

```
(define plusk
        (let ((x 4))
                (lambda (k) (+ x k))
        )
)
```

What is the value of (plusk 10)?

Example of let:

```
(define plusk
      (let ((x 4))
            (lambda (k) (+ x k))
      )
)
```

What is the value of (plusk 10)?

(plusk 10) = (+ 4 10) = 4 + 10 = 14

Example of let*:

```
(define plusz
      (let* ((x 4)
             (y (+ 2 x)))
           (lambda (z) (+ y z))
      )
)
```

What is the value of (plusz 10)?

Example of let*:

```
(define plusz
	(let* ((x 4)
		  (y (+ 2 x)))
		(lambda (z) (+ y z))
	)
)
```

What is the value of (plusz 10)?

(plusz 10) = (+ 6 10) = 6 + 10 = 16

Example of letrec:

```
(define plusfunc
      (lambda (v)
            (letrec ((x v)
                       (plusk (lambda (k w)
                                    (if (eq? k 0) w (plusk (- k 1) (+ w x)))
                               )
                       ))
                  plusk
            )))
```

What is (plusfunc 10)?

Example of letrec:

```
(define plusfunc
        (lambda (v)
                (letrec ((x v)
                         (plusk (lambda (k w)
                                    (if (eq? k 0) w (plusk (- k 1) (+ w x)))
                                )
                         ))
                        plusk
                )))
```

What is (plusfunc 10)?

The function plusk with x = 10: (lambda (k w) (if (eq? k 0) w (plusk (- k 1) (+ w 10))))

Example of letrec:

```
(define plusfunc
    (lambda (v)
        (letrec ((x v)
                 (plusk (lambda (k w)
                         (if (eq? k 0) w (plusk (- k 1) (+ w x)))
                        )
                 ))
            plusk
        )))
```

What is ((plusfunc 10) 4 0)?

(plusk 4 0) with x = 10: This equals 40

Topics Today

Fixed-point Combinator

Project 2

- Review of Scheme
- map, assoc, let, let*, letrec
- FIRST & FOLLOW sets algorithms
- Advice/Techniques

Project 2: FIRST/FOLLOW sets algorithms

FIRST sets algorithm: where P is the set of all production rules in the grammar

```
while (FIRST sets are still changing) do
    for each p ∈ P, of the form X → Y_1 Y_2 … Y_k do   (Is also X ::= Y_1 Y_2 ...)
        temp ← FIRST(Y_1) - {Ɛ}
        i ← 1
        while (i ≤ k - 1 and Ɛ ∈ FIRST(Y_i))
            temp ← temp U (FIRST(Y_i+1) - {Ɛ})
            i ← i + 1
        end // while loop
        if i == k and Ɛ ∈ FIRST(Y_k)
            then temp ← temp U {Ɛ}
        end // if-then
        FIRST(X) ← FIRST(X) U temp
    end // for loop
end // while loop
```

Project 2: FIRST / FOLLOW sets algorithms

FOLLOW sets algorithm: where P is the set of all production rules in the grammar

```
while (FOLLOW sets are still changing) do
    for each p ∈P, of the form A → B_1 B_2 … B_k do (Is also A ::= B_1 B_2 ...)
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if B_i ∈ NT then
                FOLLOW(B_i) ← FOLLOW(B_i) U TRAILER
                if Ɛ ∈ FIRST(B_i)
                    TRAILER ← TRAILER U (FIRST(B_i) - {Ɛ})
                else TRAILER ← FIRST(B_i)
            else TRAILER ← {B_i}
```

Topics Today

Fixed-point Combinator

Project 2

- Review of Scheme
- map, assoc, let, let*, letrec
- FIRST & FOLLOW sets algorithms
- Advice/Techniques

Project 2: Advice/Techniques

Advice/Techniques

- Start early!
- Do some examples involving map, assoc, let, let*, letrec
- Understand the Scheme structure of the grammar. (Check proj2.ss)
- Understand how the FIRST/FOLLOW set algorithms work
- Start with the utility functions
- Come to office hours if you need help.