Discussion 09

Interpreters, Tail Calls, Scheme Data Abstractions (7/27)

Antonio Kam

`anto [at] berkeley [dot] edu`

All slides can be found on

`teaching.rouxl.es`

Announcements

- Scheme Project gets released tomorrow
- The HW for next week will be on Scheme + Scheme Data Abstractions, so just get ready for that!

Notes from last section

- good disc today/nice work
  - 🥺 ty
- I appreciate how you help students by guiding them with questions instead of telling them the answer
  - this is the sort of thing i try and go for even during lab/oh - if you haven't noticed by now, i tend to ask questions in response to other questions (and if you haven't, now you know 👀)
- Cats or Dogs?
  - cats!
  - i do like dogs also but oh man cats :3

Notes from last section

- found u: https://arc.net/credits 👀
  - btw if you use osx i highly recommend switching over to arc as your main browser
  - it's very good (it's what i personally have been using for a while)
- Do you have any favorite memes?
  - not that i can think of off the top of my head, but you will find a few good ones during the sql lecture i'm giving lol
- In the game Undertale, sometimes a rare theme plays while fighting Sans in the Genocide route. It's a rather delicate tune. Additionally, Sans is one of two characters within the game that is aware that the player can reset the game, the only other one being the final bosses of the pacifist and normal route
  - i think i've heard the alternate song, and it's kinda 🤯 - very interesting composition/arrangement

Temperature Check 🌡️

- Interpreters
- Tail Calls
- Scheme Data Abstractions

Interpreters

What is an interpreter

- An interpreter is a program used to understand other programs
- We will start off with the Calculator language to look at Scheme Syntax (and how to interpret it)
- You will be writing an interpreter for the final project (this is a super cool project)
- note to anto: open discussion worksheet

How does it work

- Essentially is the code version of what we've studied during week 1!
- Evaluate operators/operands, apply operands to operators; this whole process is done through our own interpreter

`Pair` Class

- We will be using the `Pair` class (very similar to `Link`, but with a few added bonuses/slight differences)
    - Has a `first` and `rest` attribute
    - Also has a `map` method where it applies a function to every argument
    - Must provide `nil`, does not default to `Link.empty` like with our `Link` class
- If `p` is a `Pair` containing a proper call expression, we get the operator by doing `p.first`, and get the operands with `p.rest`. To get the first operand, we need to do `p.rest.first`

`Pair` class

```
class Pair:
    """Represents the built-in pair data structure in Scheme."""
    def __init__(self, first, rest):
        self.first = first
        if not scheme_valid_cdrp(rest):
            raise SchemeError("cdr can only be a pair, nil, or a promise but was {}".format(rest))
        self.rest = rest

    def map(self, fn):
        """Maps fn to every element in a list, returning a new
        Pair.

        >>> Pair(1, Pair(2, Pair(3, nil))).map(lambda x: x * x)
        Pair(1, Pair(4, Pair(9, nil)))
        """
        assert isinstance(self.rest, Pair) or self.rest is nil, \
            "rest element in pair must be another pair or nil"
        return Pair(fn(self.first), self.rest.map(fn))
```

`calc_eval` and `calc_apply`

```
def calc_eval(exp):
    if isinstance(exp, Pair): # Call expressions
        return calc_apply(calc_eval(exp.first), exp.rest.map(calc_eval))
    elif exp in OPERATORS:       # Names
        return OPERATORS[exp]
    else:                        # Numbers
        return exp
```

```python
def calc_apply(operator, args):
    """Apply the named operator to a list of args.

    >>> calc_apply('+', as_scheme_list(1, 2, 3))
    6
    >>> calc_apply('-', as_scheme_list(10, 1, 2, 3))
    4
    >>> calc_apply('*', nil)
    1
    >>> calc_apply('*', as_scheme_list(1, 2, 3, 4, 5))
    120
    >>> calc_apply('/', as_scheme_list(40, 5))
    8.0
    """
    if not isinstance(operator, str):
        raise TypeError(str(operator) + ' is not a symbol')
    if operator == '+':
        return reduce(add, args, 0)
    [...]
```

Worksheet!

Tail Recursion

Tail Recursion

- Sometimes with recursive functions it's possible to write it in a **tail recursive** way.
- A **tail call** occurs when a function calls another function as the *last action* of the current frame.

Example of non-tail recursive function

```
(define (factorial n)
  (cond
    ((= n 0) 1)
    (else (* n (factorial (- n 1))))
  )
)
```

- The recursive call might be in the last line, but it isn't the last action here
- Another way to see this is through environment diagrams (I'll draw them up in a bit)

Example of tail recursive function

```
(define (factorial n)
  (define (helper n result)
    (cond
      ((= n 0) result)
      (else (helper (- n 1) (* result n)))
    )
  )
  (helper n 1)
)
```

- In this case, the last recursive call here is our helper function itself `(helper (- n 1) (* result n))`
- Tail recursive functions very often need helper functions (for additional variables)

Why use tail recursion

- You won't get a recursion depth error with a tail recursive function because only a constant number of frames are needed at any given point
- More efficient - brings recursion closer to iteration in efficiency
- Scheme will discard the frames already used - doesn't need as much space to run

Tail Context

- When trying to identify whether a function call is a tail call, we look to see whether the call expression is a tail context
    - If there are any recursive calls before the tail context, the function is not tail recursive
- Essentially, the idea is to look at the *last* expression of the body: this may come in different forms
    - The second/third 'operand' inside `if`
    - Any non-predicate body inside `cond`
    - Last 'operand' in `and` or `or`
    - Last 'operand' in `begin`
    - Last 'operand' in `let`
    - (Will be useful for implementing EC! Reference the discussion worksheet later)

Is Tail Call

```
(define (question-a x)
  (if (= x 0) 0
      (+ x (question-a (- x 1)))))
```

Is Tail Call

```
(define (question-b x y)
  (if (= x 0) y
      (question-b (- x 1) (+ y x))))
```

Is Tail Call

```
(define (question-c x y)
  (if (> x y)
      (question-c (- y 1) x)
      (question-c (+ x 10) y)))
```

Is Tail Call

```
(define (question-d n)
  (if (question-d n)
      (question-d (- n 1))
      (question-d (+ n 10))))
```

Is Tail Call

```
(define (question-e n)
  (cond ((<= n 1) 1)
        ((question-e (- n 1)) (question-e (- n 2)))
        (else (begin (print 2) (question-e (- n 3))))))
```

Worksheet!

Data Abstractions

What are data abstractions

- In Python, we used classes to describe certain objects
- Classes don't exist in scheme 🙀
- As a result, data abstractions help to fill in this gap
    - If we want something in scheme to describe objects, we use data abstractions!

Data abstractions

- Constructors
  - Similar to `__init__` in our classes
  - Function that builds our abstract data type
- Selectors
  - Functions that receive information from the data type
  - Similar to getting an instance variable value (`t.label`, `s.first`)

Example Data Abstraction (Python)

Constructor:

```python
def couple(first, second):
    return [first, second]
```

Selectors:

```python
def first(couple):
    return couple[0]

def second(couple):
    return couple[1]
```

```python
c = couple(1, 2)
c1, c2 = first(c), second(c) # c1 = 1, c2 = 2
```

Example Data Abstraction (Scheme)

Constructor:

```scheme
(define (couple first second) (list first second))
```

Selectors:

```scheme
(define (first couple) (car couple))
(define (second couple) (car (cdr couple)))
```

```scheme
(define c (couple 1 2))
(first c)
(second c)
```

Violating Data Abstraction Barriers

- One thing you cannot do when it comes to data abstractions is **violate the data abstraction barriers**.
- This basically means that you must use the constructors and selectors when dealing with data abstractions
  - This is because you do not necessarily know how the data abstractions are implemented

Example Data Abstraction (Python)

Constructor:

```python
def couple(first, second):
    return {"first": first, "second": second}
```

Selectors:

```python
def first(couple):
    return couple["first"]

def second(couple):
    return couple["second"]
```

```python
c = couple(1, 2)
c1, c2 = first(c), second(c) # c1 = 1, c2 = 2
```

City

- Let's say we have a definition for a city data abstraction
- We just need the documentation for what the names mean, but we don't need to know how it's implemented:

```
(make-city name lat lon)

(get-name city)
(get-lat city)
(get-lon city)
```

- All we need to know is that we can use these definitions

Results from last section (`links.rouxl.es/disc`)

- 8-9 (3)
- 30 minutes before first class
- 7am (how) (5)
- 6:30 am 😭
- As late as i can ✅
- 10 (2)

Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!