Abstract Data Types

David E. Culler
CS8 – Computational Structures in Data Science
http://inst.eecs.berkeley.edu/~cs88

Lecture 7
Oct 8, 2018

Computational Concepts Toolbox

- Data type: values, literals, operations, e.g., int, float, string
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
  - indexing
- Data structures
  - Tuple assignment
  - Call Expressions
  - Function Definition Statement
  - Conditional Statement
- Iteration:
  - data-driven (list comprehension)
  - control-driven (for statement)
- Higher Order Functions
  - Functions as Values
  - Functions with functions as argument
  - Assignment of function values
- Recursion
- Lambda - function valued expressions

Environments and Closures

Administrative Issues

- Midterm went very well
- Project 1 is out
- Mid Term Survey Thanks

Weekly “Pipeline”

Universality

- Everything that can be computed, can be computed with what you know now.
- Well
- or poorly

Today’s Lecture

- Administrative Issues
- Review: lambda
- New Concept: Abstract Data Type
- Example Illustration: key-value store
  - Internal representation 1: list of pair
  - Internal representation 2: pair of lists (including zip intro)
- A simple application over the KV interface
- New language construct: dict
- Key-Value store 3: dict
- Optional Exercises

http://datahub.berkeley.edu/user-redirect/interact?account=data-8&repo=cs-connector&branch=gh-pages&path=ADT
Lambda Examples

```
>> sort([1,2,3,4,5], lambda x: x)  
[1, 2, 3, 4, 5]
>> sort([1,2,3,4,5], lambda x: -x) 
[5, 4, 3, 2, 1]
>> sort([(2, 'hi'), (1, 'how'), (5, 'goes'), (7, 'I')], 
   lambda x:0) 
[(1, 'how'), (2, 'hi'), (5, 'goes'), (7, 'I')]
>> sort([(2, 'hi'), (1, 'how'), (5, 'goes'), (7, 'I')], 
   lambda x:1) 
[(7, 'I'), (5, 'goes'), (2, 'hi'), (1, 'how')]
>> sort([(2,'hi'),(1,'how'),(5,'goes'),(7,'I')], 
   lambda x: len(x[1])) 
[(7, 'I'), (2, 'hi'), (1, 'how'), (5, 'goes')]
```

Abstract Data Type

- **Constructions:**
  - Internal Representation
  - Implementation on that Internal representation

Examples Data Types You have seen

- **Lists**
  - Constructors:
    - `list( ... )`
    - `[]`, `<exp>`, `...`
  - Selectors: `<var> in <list>`
  - Operations: `in, not in, +, *`, `len, min, max`
    - `Mutable` ones too (but not yet)

- **Tuples**
  - Constructors:
    - `tuple( ... )`
    - `()`
  - Selectors: `<tuple> [ <index or slice> ]`
  - Operations: `in, not in, +, *`, `len, min, max`

- **Ranges**
  - Constructors:
    - `range(<end>), range(<start>,<end>)`
  - Selectors: `range [ <index or slice> ]`
  - Operations: `in, not in, +, *`, `len, min, max`

More “Built-In” Examples

- **Lists**
- **Tuples**
- **Strings**
  - Constructors:
    - `str( ... )`
    - `'<chars>', 'chars>'`
  - Selectors: `str [ <index or slice> ]`
  - Operations: `in, not in, +, *`, `len, min, max`

- **Ranges**
  - Constructors:
    - `range(<end>), range(<start>,<end>)`
  - Selectors: `range [ <index or slice> ]`
  - Operations: `in, not in, +, *`, `len, min, max`

A New Abstract Data Type: Key-Value

- **Collection of key-Value bindings**
  - Key: Value
- **Many real-world examples**
  - Dictionary, Directory, Phone book, Course Schedule, Facebook Friends, Movie listings,...

Given some Key, What is the value associated with it?
Key-Value ADT

- **Constructors**
  - `kv_empty`: create an empty KV
  - `kv_add`: add a key:value binding to a KV
  - `kv_create`: create a KV from a list of key:value tuples

- **Selectors**
  - `kv_items`: list of (key, value) tuple in KV
  - `kv_keys`: list of keys in KV
  - `kv_values`: list of values in KV

- **Operations**
  - `kv_len`: number of bindings
  - `kv_in`: presence of a binding with a key
  - `kv_display`: external representation of KV

---

A Layered Design Process

- Build the application based entirely on the ADT interface
  - Operations, Constructors and Selectors
- Build the operations in ADT on Constructors and Selectors
  - Not the implementation representation
- Build the constructors and selectors on some concrete representation

---

Example 1

- KV represented as list of (key, value) pairs

---

Example 2

- KV represented as pair of lists – (keys, values)

---

zip

- Zip (like a zipper) together k lists to form a list of k-tuples
Dictionaries

• Lists, Tuples, Strings, Range
• Dictionaries
  - Constructors:
    » dict( <list of 2-tuples> )
    » dict( <key>=<val>, ... ) # like kwargs
    » { <key>:<val>, ... }
    » { <key>=<val> for <iteration expression> }
  >>> { x:y for x,y in zip(['a','b'],[1,2]) } 
    » {'a': 1, 'b': 2}
  - Selectors: <dict> [ <key> ]
    » <dict>.keys(), .items(), .values()
    » <dict>.get(key [, default] )
  - Operations:
    » Key in, not in, len, min, max
    » <dict>[ <key> ] = <val>

Beware

• Built-in data type dict relies on mutation
  » Clobbers the object, rather than "functional" – creating a new one
• Throws an errors of key is not present
• We will learn about mutation shortly

Building Apps over KV ADT

friend_data = [
  ('Christine Strauch', 'Jack Chow'),
  ('Christine Strauch', 'Lydia Lu'),
  ('Jack Chow', 'Christine Strauch'),
  ('Casey Casem', 'Christine Strauch'),
  ('Casey Casem', 'Jack Chow'),
  ('Casey Casem', 'Frances Catal Buloan'),
  ('Casey Casem', 'Joy De Rosario'),
  ('Casey Casem', 'Casey Casem'),
  ('Frances Catal Buloan', 'Jack Chow'),
  ('Jack Chow', 'Frances Catal Buloan'),
  ('Joy De Rosario', 'Lydia Lu'),
  ('Joy De Lydia', 'Jack Chow')
]

• Construct a table of the friend list for each person

Example 3

• KV represented as dict

Example: make_friends

```python
def make_friends(friendships):
    friends = kv_empty()
    for (der, dee) in friendships:
        if not kv_in(friends, der):
            friends = kv_add(friends, der, [dee])
        else:
            der_friends = kv_get(friends, der)
            friends = kv_add(kv_delete(friends, der),
                             der, [dee] + der_friends)
    return friends
```
C.O.R.E concepts

1. **Compute**
   - Perform useful computations treating objects abstractly as whole values and operating on them.

2. **Operations**
   - Provide operations on the abstract components that allow ease of use – independent of concrete representation.
   - Constructors and selectors that provide an abstract interface to a concrete representation.

3. **Representation**
   - Execution on a computing machine.

4. **Evaluation**
   - Abstraction Barrier

Creating an Abstract Data Type

- **Constructors & Selectors**
  - Express the behavior of objects, invariants, etc.
  - Implemented (abstractly) in terms of Constructors and Selectors for the object.

- **Representation**
  - Implement the structure of the object.

- **An abstraction barrier violation** occurs when a part of the program that can use the higher level functions uses lower level ones instead.
  - At either layer of abstraction.

- **Abstraction barriers make programs easier to get right, maintain, and modify**.
  - Few changes when representation changes.

Exercises

- **Read 2.2**, reread 2.3, esp 2.3.6
- **Modify all three KV ADTs to avoid ever adding duplicate keys**
- **Create and ADT for a shopping cart containing a collection of products and their order count**
  - `cart()` – creates an empty cart
  - `cart_add(cart, product)` – returns a new cart that includes an additional order of product, or the first one
  - `cart_print(cart)` – prints the contents of the cart
  - `cart_products(cart)` – returns the list of products ordered
  - `cart_items(cart)` – returns list of (product, count)
  - `cart_remove(cart, product)` – returns a new cart with product removed
- **Create an 1D array abstraction (like np.array)** using lists as representation.