Programming Paradigms

Control Flow (Part 4)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Summer 2020
Overview

- Expression Evaluation
- Structured and Unstructured Control Flow
- Selection
- Iteration
- Recursion
Iteration

- Essential language construct
  - Otherwise: Amount of work done is linear to program size

- Two basic forms of loops
  - Enumeration-controlled:
    Once per value in finite set
  - Logically controlled:
    Until Boolean expression is false
Enumeration-controlled Loops

- Most simple form: Triple of
  - Initial value
  - Bound
  - Step size

**Fortran 90:**
```fortran
do i = 1, 10, 2
  ...
enddo;
```

**Modula-2:**
```fortran
FOR i := 1 TO 10 BY 2 DO
  ...
END
```
Enumeration-controlled Loops

- **Most simple form: Triple of**
  - Initial value
  - Bound
  - Step size

**Fortran 90:**
```fortran
do i = 1, 10, 2
  ...
enddo;
```

**Modula-2:**
```modula-2```
```FOR i := 1 TO 10 BY 2 DO
  ...
END```

*Iterations with $i = 1, 3, 5, 7, 9$*
Semantic Variants

Different PLs offer different variants

■ Can you leave the loop in the middle?
■ Can you modify the loop variable?
■ Can you modify the values used to compute the loop bounds?
■ Can you read the loop variable in/after the loop?
Iterators

- Special enumeration-controlled loop: *Iterates through any kind of set/sequence of values*
  - E.g., nodes of a tree or elements of a collection

- **Decouples two algorithms**
  - How to *enumerate* the values
  - How to *use* the values

- **Three flavors**
  - “True” iterators, iterator objects, first-class functions
“True” Iterators

- Subroutine with `yield` statements
  - Each `yield` “returns” another element
- Popular, e.g., in Python, Ruby, and C#
- Used in a `for` loop
  - Example (Python):
    ```python
    # range is a built-in iterator
    for i in range(first, last, step):
        ...
    ```
Example: Binary Tree

class BinTree:
    def __init__(self, data):
        self.data = data
        self.lchild = self.rchild = None

    # other methods: insert, delete, lookup, ...

    def preorder(self):
        if self.data is not None:
            yield self.data
        if self.lchild is not None:
            for d in self.lchild.preorder():
                yield d
        if self.rchild is not None:
            for d in self.rchild.preorder():
                yield d
Iterator Objects

- Regular object with **methods** for
  - Initialization
  - Generation of **next value**
  - Test for completion

- Popular, e.g., in Java and C++

- Used in **for loop**

```java
for (Iterator i = c.iterator(); i.hasNext(); ) {
    ... = i.next();
}
```
Iterator Objects

- Regular object with **methods** for
  - Initialization
  - Generation of *next value*
  - Test for completion

- Popular, e.g., in Java and C++
- Used in *for* loop

```java
for (Iterator i = c.iterator(); i.hasNext(); ) {
    ... = i.next();
}
```
Iterator Objects

- Regular object with methods for
  - Initialization
  - Generation of next value
  - Test for completion
- Popular, e.g., in Java and C++
- Used in for loop

```java
for (Iterator i = c.iterator(); i.hasNext(); ) {
    ... = i.next();
}
```

Since Java 5

```java
for (Element e : c) {
    ...
}
```
class BinTree<T> implements Iterable<T> {
    BinTree<T> left; BinTree<T> right; T val;

    // other methods: insert, delete, lookup

    public Iterator<T> iterator() {
        return new TreeIterator(this);
    }

    private class TreeIterator implements Iterator<T> {
        public boolean hasNext() {
            ... // check if there is another element
        }
        public T next() {
            ... // return the next element
        }
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
Iterating with First-Class Functions

- Two functions
  - One function about **what to do** for each element
  - Another function that **calls** the first function for each element

- Example (Scheme):

```scheme
(define uptoby
 (lambda (low high step f)
  (if (<= low high)
      (begin
       (f low)
       (uptoby (+ low step) high step f))
      '()))
)
```
Iterating with First-Class Functions

- **Two functions**
  - One function about **what to do for each element**
  - Another function that **calls** the first function for each element

- **Example (Scheme):**

```
(define uptoby
  (lambda (low high step f)
    (if (<= low high)
      (begin
        (f low)
        (uptoby (+ low step) high step f))
      '())))
```
Iterating with First-Class Functions

- Two functions
  - One function about what to do for each element
  - Another function that calls the first function for each element

- Example (Scheme):

```
(define uptoby
  (lambda (low high step f)
    (if (<= low high)
        (begin
          (f low)
          (uptoby (+ low step) high step f))
        '())))
```

Defines a function with four arguments.
Iterating with First-Class Functions

- Two functions
  - One function about what to do for each element
  - Another function that calls the first function for each element

- Example (Scheme):

```
(define upto
(lambda (low high step f)
(if (<= low high)
  (begin
    (f low)
    (uptoby (+ low step) high step f))
  '())))
```

Calls $f$ with the next element
Iterating with First-Class Functions

- **Two functions**
  - One function about what to do for each element
  - Another function that calls the first function for each element

- **Example (Scheme):**

```scheme
(define uptoby
  (lambda (low high step f)
    (if (<= low high)
        (begin
          (f low)
          (uptoby (+ low step) high step f))
        '()))
)
```

Recursively calls `uptoby` to handle the remaining elements
Iterating with First-Class Functions (2)

- Originally, proposed in functional languages
- Nowadays, available in many modern PLs through libraries
  - E.g., Java
    ```java
    mySet.stream().filter(e -> e.someProp > 5)
    ```
  - E.g., JavaScript
    ```javascript
    myArray.filter(e => e.someProp > 5)
    ```
Iterating with First-Class Functions (2)

- Originally, proposed in *functional languages*
- Nowadays, *available* in many modern PLs through *libraries*
  - E.g., Java
    ```java
    mySet.stream().filter(e -> e.someProp > 5)
    ```
    Iterates through all elements and returns a filtered subset
  - E.g., JavaScript
    ```javascript
    myArray.filter(e => e.someProp > 5)
    ```
Iterating with First-Class Functions (2)

- Originally, proposed in functional languages
- Nowadays, available in many modern PLs through libraries
  - E.g., Java
    ```java
    mySet.stream().filter(e -> e.someProp > 5)
    ```
  - E.g., JavaScript
    ```javascript
    myArray.filter(e => e.someProp > 5)
    ```

  Boolean function that decides which elements to keep
Logically Controlled Loops

Whether to **continue to iterate** decided through a **Boolean expression**

- **Pre-test:**  
  ```
  while (cond) {
      ...
  }
  ```

- **Mid-test:**  
  ```
  for (;;;)
  {
      ...
      if (cond) break
  }
  ```

- **Post-test:**  
  ```
  do {
      ...
  } while (cond)
  ```
Quiz: Iteration

Which of the following statements is true?

- Iterators are a form of logically controlled loops.
- A “true” iterator yields one element each time it is called.
- Iterator objects have a method that yields another element each time it is called.
- Iterating with first-class functions does not require a for-loop.

Please vote in Ilias
Quiz: Iteration

Which of the following statements is true?

■ Iterators are a form of logically controlled loops.
■ A “true” iterator yields one element each time it is called.
■ Iterator objects have a method that yields another element each time it is called.
■ Iterating with first-class functions does not require a for-loop.

Please vote in Ilias