Vectors and Iteration

Corky Cartwright
Vivek Sarkar
Department of Computer Science
Rice University

Outline

- Vectors in Scheme
- Functional vs. Imperative views of
  - Iteration
  - Arrays
- Today’s lecture is all bonus material!
  - Will not be covered in test or homework

A First Look at Vectors (Section 29.3)

Goal: array-like data structure with O(1) lookup time for a given index

- Vector creation
  - `(vector V-0 ... V-n)` creates a vector with n+1 elements, `V-0` through `V-n`
  - `(build-vector n f)` creates a vector with n elements, `(f 0)` through `(f (- n 1))`
    - Simple case of an *array comprehension*

Vector Operations (contd)

- `(vector-length V)` returns the number of items in vector V
  - Results in an error if V is not a vector
- `(vector-ref V i)` returns the ith item in vector V
  - Results in an error if V is not a vector or i is not a number or i < 0 or i >= (vector-length V)
- `(vector? V)` returns true if V is a vector

Simple example: sum-of-3

```
;; vector-sum-of-3 :
;;  (vector number number number)-> number
;; Return sum of first three items of vector
(define (vector-sum-of-3 v)
    (+ (vector-ref v 0)
       (vector-ref v 1)
       (vector-ref v 2)))
```

- Example: **`(vector-sum-of-3 (vector 2 4 6 8 10))`**
- NOTE: vector is like cons, and vector-ref is like first/rest

Binary Search on a Sorted Vector of Numbers

```
;; bin-srch: asvon number number number -> number
;; For input vector V, value X, lower & upper bounds
;; lo and hi, return index i in lo … hi such that
;; (vector-ref V i) = X, else return -1 if X not found
;; NOTE: use Advanced Student setting to use vectors
(define (bin-srch V X lo hi)
  (let ((mid (floor (/ (+ lo hi) 2))))
      (cond
        ((> lo hi) -1)
        ((= (vector-ref V mid) X) mid)
        ((> (vector-ref V mid) X) (bin-srch V X lo(- mid 1)))
        ((< (vector-ref V mid) X) (bin-srch V X (+ mid 1) hi)))
  ))
```

Execution Time Complexity

- What is the execution time complexity of binary search using a vector?

- How would the complexity of binary search change if we replaced the vector by a list of pairs (and used list-ref instead of vector-ref)?

Vectors vs. lists

- Pro: vector-ref can be used to access any element in a vector in O(1) time
  - Multiple first/rest operations may be needed to traverse a list
- Con: extending a vector or extracting from a vector takes O(n) time
  - Constructing a list with a new element at the start of an existing list takes O(1) time (cons)
  - Extracting the tail of a list takes O(1) time (rest)

Iteration

- Iterating over a vector/list in a functional language is usually accomplished by (tail) recursion
- Iterating over a vector/list in an imperative language is usually accomplished by iteration
  - e.g., while-loops and for-loops in Java
- Does this mean that iteration is inherently non-functional?

Sisal: Example of a Functional Language with Iteration

- Sisal stands for Streams and Iteration in a Single Assignment Language
- Defined in 1983, revised and frozen in 1985
- Original collaborators were LLNL, Colorado State U, University of Manchester, and DEC
  - Used for research at many other institutions, including Stanford University
- Language design strongly influenced by dataflow computation model

Sisal Objectives

- to define a general-purpose functional language
- to define a language independent intermediate form for dataflow graphs
- to develop optimization techniques for high performance parallel applicative computing
- to develop a microtasking environment that supports dataflow on conventional computer systems
- to achieve execution performance comparable to imperative languages
- to validate the functional style of programming for large-scale scientific applications

Some Simple Sisal Programs

```
% Hello world!
define main
function main(returns array[character])
  “hello world”
end function
```

```
% Simple arrays
define main
function main(A: array[integer] returns integer, array[integer])
  for element in A       % parallel loop with independent iterations
      sqr := element * element
 returns
      value of sum sqr   % reduce operation
      array of sqr       % array comprehension, like vector-build
 end for
end function
```

Sequential iteration with for-initial loop expressions

- Not all loops are implicitly data parallel
- Sisal supports an iterative form that supports the idea of “loop carried dependencies”
- The loop body is allowed to reference both the “new” and the “old” value of a definition (variable)
- An separate body defines the initial values

Example #1: Iterated Function Composition

```
for initial   % Initializer body is like the zeroth iteration
        i := 0;
        accum := 0;
while i < n repeat
        i := old i + 1;             % Note the use of "old" to denote previous value
        accum := f(old accum);
returns
       value of accum
end for
```

Scheme equivalent:

```
(local
  ((define (g i accum)
    (cond [(i < n) (g (+ i 1) (f accum))] [else accum])))
    (g 0 0))
```

Example #2: 3-point stencil w/ Array Replace Operation

```
for initial
        A := some_value();           % This is the zeroth “iteration”
         i := array_liml(A);          % Lower bound of array A’s indices
while i < array_limh(A) repeat
         i := old i + 1;
         A := old A[i:   ( old A[i-1] + old A[i] + old A[i+1]) / 3.0   ];
returns
       value of A
end for
```

- Array replace operation --- A[i : X] returns a new array A’ identical to A, except that element I is replaced by X
    - Functional alternative to A[i] = X; in Java or C
- Semantically, A’ is a copy of A, but implementations try to make their best effort to eliminate as many copies as possible.

Announcements

- Midterm to be distributed on Friday (Feb 19th)