CS 2SC3 and SE 2S03 Fall 2009

04 Basic Data Structures

William M. Farmer

Department of Computing and Software
McMaster University

5 November 2009

Records

- A record is a data structure that holds a tuple whose members are named.
- The members of a record are called fields and are indexed by their names.
- Records are also called structures (as in C).
- The type of a record is declared before a record is defined.
- The fields of a record are usually mutable.
- Records of a type t have a constructor, an indexed selector, and possibly an indexed mutator.

Records in OCaml

- Record type declaration:

  `type` rec-type `=` $\{\text{name}_1$ `:` t_1 `;` $\ldots$ `;` name_n `:` $t_n\}$ `;;`

- Constructor:

  $\{\text{name}_1$ `=` expr_1 `;` $\ldots$ `;` name_n `=` $\text{expr}_n\}$

- Selector:

  e `.` n

  where e is an expression of a record type having a field named n.

- A field in a record is mutable if it is declared mutable:

  `type` rec-type `=` `{` $\ldots$ `;` `mutable` name_i `:` t_i `;` $\ldots$ `}` `;;`

- Mutator:

  e_1 `.` n `<-` e_2

  where e_1 is an expression of a record type having a mutable field named n and e_2 is of the same type as e_1 `.` n.

Arrays

- An array is a data structure that holds a finite sequence of values such that each element of the sequence can be directly accessed and modified.
- The members of an array are called cells and are indexed by natural numbers.
- Arrays usually have a constructor, an indexed selector, and an indexed mutator.
- Arrays may be multidimensional.
  - A one-dimensional array is called a vector (as in OCaml).
- Strings are often implemented as arrays (as in C and OCaml).

Arrays in OCaml

- Constructor:

  $[|\text{expr}_0; \ldots; \text{expr}_{n-1}|]$

  constructs an array of type t `array` and length n where t is the type of $\text{expr}_0, \ldots, \text{expr}_{n-1}$.

- Selector:

  $e_1.(e_2)$

  where e_1 is an expression denoting a array and e_2 is an expression denoting a value i of type `int` with $0 \leq i \leq n-1$ where n is the length of the array.

- Mutator:

  $e_1.(e_2) \texttt{<-} e_3$

- The `Array` module contains several other constructors, selectors, and mutators including `create`, `append`, and `length`.

Character Strings in OCaml

- Strings are a special kind of array in OCaml.
- Constructor:

  $"c_1 c_2 \cdots c_n"$

- Selector:

  $e_1 . [e_2]$

- Mutator:

  $e_1 . [e_2]$ <- e_3

Stacks

- A stack is a data structure that holds a finite sequence of values such that the elements of the sequence are accessed according to the principle of last in first out (LIFO).
- Stacks are employed extensively in computer systems.
- Constructor:

  $$\text{bottom} : \text{void} \longrightarrow \text{stack}.$$

- Selectors:

  $$\text{height} : \text{stack} \longrightarrow \text{nat}.$$
  $$\text{top} : \text{stack} \longrightarrow \text{element}.$$

- Mutators:

  $$\text{push} : \text{element}, \text{stack} \longrightarrow \text{void}.$$
  $$\text{pop} : \text{stack} \longrightarrow \text{void}.$$

Queues

- A queue is a data structure that holds a finite sequence of values such that the elements of the sequence are accessed according to the principle of first in first out (FIFO).
- Constructor:

  $\text{empty} : \text{void} \longrightarrow \text{queue}.$

- Selectors:

  $\text{length} : \text{queue} \longrightarrow \text{nat}.$
  $\text{front} : \text{queue} \longrightarrow \text{element}.$

- Mutators:

  $\text{push} : \text{element}, \text{queue} \longrightarrow \text{void}.$
  $\text{pop} : \text{queue} \longrightarrow \text{void}.$

Enumerated Types and Disjoint Sum Types

- An enumerated type consists of a finite set of named values.
  - The names of the values behave like constants.
- A disjoint union type is the union of a set of tagged types.
  - The tags serve as constructors for the values of the type.
  - The tags make the component types disjoint.
- An enumerated type is a disjoint union of unit types.

Algebraic Data Types

- An algebraic data type consists of a set of values defined by a set of constructors.
- Several types are special cases of algebraic data types:
  - Product type (one n-ary constructor with $n \geq 1$).
  - Enumerated type (several 0-ary constructors).
  - Disjoint union type (several unary constructors).
- An inductive data type is a recursively defined algebraic data type.
  - Example 1: Type of lists defined by the constructors [] and cons.
  - Example 2: Type of unary natural numbers defined by the constructors 0 and S.
  - Example 3: Type of stacks defined by the constructors bottom and push.

Abstract Data Types

- An abstract data type (ADT) is a set of data and a set of operations that can be applied to the data.
  - The data and operations are described abstractly without reference to how they are implemented.
- ADTs are essentially the same as algebras in algebraic specification and mathematical structures in mathematics.
- An algebraic data type is a special case of an ADT in which the operations are unconstrained.
- An ADT is a special case of a module that consists of an interface and an implementation.
- ADTs support:
  - The Principle of Separation of Concerns: the interface of an ADT is separated from its implementation.
  - The Principle of Information Hiding: the "secret" of the ADT is hidden from the user.

Sum Types in OCaml

- A sum type in OCaml represents an algebraic data type.
  - Sum types are also called union types and variant types.
  - Sum types can represent enumerated types and disjoint union types.
- Sum type declaration:

```
type name =
    Name_1 [of t_1]
  | Name_2 [of t_2]
      ⋮
  | Name_n [of t_n] ;;
```

- The Name_i are constructors that:
  1. Construct values of the sum type.
  2. Tag the values of the sum type to distinguish the components of the corresponding disjoint union.
  3. Select values of the sum type via pattern matching.

Summary of Types in OCaml

- Types of Immutable Values
  - Basic types: `unit`, `bool`, `int`, `float`, `char`.
  - Function types: t_1 `->` t_2.
  - Product types: t_1 `*` $\cdots$ `*` t_n.
  - List types: t `list`.
  - Sum types: Name_1 [`of` t_1] `|` $\ldots$ `|` Name_n [`of` t_n].
- Types of Mutable Values
  - Reference types: t `ref`.
  - Record types:
    $\{$[`mutable`] name_1 `:` t_1`;` $\ldots$`;` [`mutable`] name_n `:` $t_n\}$.
  - Array types: t `array`.
  - String type: `string`.

Type Declarations in OCaml

- (Recursive) type declarations:

  `type` name `=` type_expression `;;`

- Mutually recursive type declarations:

  `type` name_1 `=` type_expression_1
  `and` name_2 `=` type_expression_2
  $\vdots$
  `and` name_n `=` type_expression_n `;;`

- Parameterized type declarations:

  `type` 'a name `=` type_expression `;;`
  `type` ($'a_1, \ldots, 'a_n$) name `=` type_expression `;;`

What are Dynamic Data Structures?

- A static data structure is a data structure whose size is fixed during run time.
  - Overhead cost is low.
  - Expansion cost is high.
- A dynamic data structure is a data structure whose size changes during run time.
  - Overhead cost is high.
  - Expansion cost is low.

Linked Lists

- A linked list is a dynamic data structure consisting of a finite sequence of linked nodes that holds a finite sequence of values.
- Each node is a record of type t containing various data fields and one or two references of type t:
  1. A field named next that points to the "next" node.
  2. A field named previous that points to the "previous" node.

  Note: The type t is self-referential.
- A link list has four basic forms:
  1. Singly linked (having one of the next and previous fields).
  2. Doubly linked (having both of the next and previous fields).
  3. Singly linked in a circle.
  4. Doubly linked in a circle.

Linked List Operators

- Constructor: Starts the construction of a new linked list.
- Node selector: A node is accessed sequentially by following the links until the node is reached.
- Node mutators:
  - Node removal: A node is removed from the linked list.
  - Node insertion. A new node is inserted into the linked list.

Comparison of Linked Lists with Arrays

- Linked lists are more space efficient than arrays.
  - Arrays fill up; linked lists do not.
  - Arrays may have empty cells; linked lists do not.
  - Resizing an array is costly; resizing a linked list is not.
- Access to an array element is faster than to linked list elements.
  - Arrays support random access.
  - Linked lists support only sequential access.
- Linked lists have higher space and time overhead than arrays.
  - Cells requires more space to hold node references.
  - Removing and inserting elements takes more time.

Binary Trees

- A binary tree is a dynamic data structure consisting of a tree of linked nodes that holds a tree of elements.
- Each node is a record of type t containing various data fields and two references of type t:
  1. A field named left that points to the "left" child node.
  2. A field named right that points to the "right" child node.
- Data can be stored at each node or at only the leaf nodes.