Datatypes in ML

lists, tuples, arrays, records, variants …

Walter Cazzola
Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it

OCaML’s Primitive Datatypes

Introduction

Even if not explicitly said

- ML is a strongly and statically-typed programming language,
- the type of each expression is inferred from the use.

OCaML’s Primitive Datatypes

Booleans

OCaML provides two constants
- true and false

Operations on Booleans

<table>
<thead>
<tr>
<th>logic operators</th>
<th>logical and, or, and negation respectively</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;,</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>relational operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>==, &lt;&gt;</td>
</tr>
<tr>
<td>less than, greater than, less than or equal to and greater than or equal to operators</td>
</tr>
</tbody>
</table>

OCaML’s Primitive Datatypes

Strings

- they are native in OCaML
- several operations come from the String module

Operations on Strings

<table>
<thead>
<tr>
<th>basic operators</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-[]</td>
<td>string concatenation</td>
</tr>
<tr>
<td>s.[n] &lt;- c</td>
<td>positional access to chars</td>
</tr>
</tbody>
</table>

| changes the nth char of s in c |

```ocaml
let s1 = "walter" and s2 = "cazzola" ;;
val s1 : string = "walter"
val s2 : string = "cazzola"
# let s = s1 ^ " " ^ s2 ;;
val s : string = "walter cazzola"
# s.[9] ;;
- : char = 'z'
# String.length(s) ;;
- : int = 14
# s.[0] <- 'W'; s.[7] <- 'C';;
- : unit = ()
# s ;;
- : string = "Walter Cazzola"
```
OCaML's Collections

Lists

- homogeneous
- cons operator ::
- concatenation operator @ (inefficient).

More operations come from the List module.

OCaML's Collections

Tuples

- fixed-length heterogeneous lists

More operations come from the Array module.

OCaML's Collections

Arrays

- direct-accessible, homogeneous, and mutable lists

More operations come from the Array module.
OCaML's Collections

Records

Records are
- name accessible (through field names),
- heterogeneous, and
- mutable (through the mutable keyword) tuples.

```ocaml
# type person = { name : string; age : int; };;
type person = { name : string; age : int; }
# let p = { name = "Walter"; age = 35 };;
val p : person = { name = "Walter"; age = 35 }
# p.name;;
- : string = "Walter"
# p.age <- p.age + 1;;
- : unit = ()
# p;;
- : person = { name = "Walter"; age = 36 }
# p.name <- "Walter Cazzola";;
Error: The record field label name is not mutable
```

User Defined Datatype in OCaML

Variants

Mutually recursive type must be declared by using the and keyword.

```ocaml
type card = Card of regular | Joker
and regular = { suit : card_suit; name : card_name; }
and card_suit = Heart | Club | Spade | Diamond
and card_name = Ace | King | Queen | Jack | Simple of int;;
let value = function
  | Joker -> 0
  | Card { name = Ace } -> 11
  | Card { name = King } -> 10
  | Card { name = Queen } -> 9
  | Card { name = Jack } -> 8
  | Card { name = Simple n } -> n ;;
```

This code defines 4 types.
- the value function gives a value to each card.

```ocaml
# type "cards.ml";;
type card = Card of regular | Joker
and regular = { suit : card_suit; name : card_name; }
and card_suit = Heart | Club | Spade | Diamond
and card_name = Ace | King | Queen | Jack | Simple of int
and value = card -> int =
  | Card { suit = Heart; name = Jack } ;;
  | . . .
```

User Defined Datatype in OCaML

Variants

Compared to OO programming,
- a variant type is equivalent to a class hierarchy composed of an abstract base class or interface representing the type and derived classes representing each of the variant type constructors.

Moreover, it is possible to manipulate them by pattern matching.

```ocaml
# use "state.ml";;
val turn : state -> state = <fun>
# let s = Off;;
val s : state = Off
# turn s;;
- : state = On
```

User Defined Datatype in OCaML

Variants

A new name can be associated to any type, both primitive and user-defined.

Variants

A variant type lists all possible shapes for values of that type.
- Each case is identified by a capitalized name, called a constructor.

```ocaml
# type int_option = Nothing | AnInteger of int;;
type int_option = Nothing | AnInteger of int
# Nothing;;
- : int_option = Nothing
# AnInteger 7;;
- : int_option = AnInteger 7
```

User Defined Datatype in OCaML

Variants

Aliasing

The easiest way to define a new type is to give a new name to an existing type.

```ocaml
# type int_pair = int*int;;
type int_pair = int * int
# let a : int_pair = (1,3);;
val a : int_pair = (1, 3)
# fst a;;
- : int = 1
```

A new name can be associated to any type, both primitive and user-defined.

Variants

A variant type lists all possible shapes for values of that type.
- Each case is identified by a capitalized name, called a constructor.

```ocaml
# type card = Card of regular | Joker
and regular = { suit : card_suit; name : card_name; }
and card_suit = Heart | Club | Spade | Diamond
and card_name = Ace | King | Queen | Jack | Simple of int
and value = function
  | Joker -> 0
  | Card { name = Ace } -> 11
  | Card { name = King } -> 10
  | Card { name = Queen } -> 9
  | Card { name = Jack } -> 8
  | Card { name = Simple n } -> n ;;
```

This code defines 4 types.
- the value function gives a value to each card.

```ocaml
# use "cards.ml";;
type card = Card of regular | Joker
and regular = { suit : card_suit; name : card_name; }
and card_suit = Heart | Club | Spade | Diamond
and card_name = Ace | King | Queen | Jack | Simple of int
and value = card -> int =
  | Card { suit = Heart; name = Jack } ;;
  | . . .
```
References

- Davide Ancona, Giovanni Lasorio, and Elena Zucca. 
  *Linguaggi di Programmazione.* 
  Città Studi Edizioni, 2007.

- Greg Michaelson. 
  *An Introduction to Functional Programming through λ-Calculus.* 
  Addison-Wesley, 1989.

- Larry C. Paulson. 
  *ML for the Working Programmer.* 