

CS421 Lecture 5a: Variant Records, Extended Example¹

Mark Hills

`mhills@cs.uiuc.edu`

University of Illinois at Urbana-Champaign

June 8, 2006

¹Based on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, and Elsa Gunter

UDT Example

Objectives

Since variant records are an important topic, this lecture supplement provides an extended example of their use. At the end of this supplement, you should:

- ▶ better understand the creation and use of variants
- ▶ know how to write functions which work over variants

The Extended Example: Poker

Card games make good examples, since they provide multiple variant types we can use (enumerations, disjoint unions) and have interesting rules we can encode. We will use poker as our example – the O'Reilly book uses French Tarot.

The Basics

- ▶ A *deck* has 52 cards
- ▶ Cards are in four suits: *hearts*, *diamonds*, *clubs*, and *spades*
- ▶ Cards are numbered from 2 to 10 and also include the *ace*, *king*, *queen*, and *jack*

Representing Suits

Suits can be represented as an *Enumeration*, since all the suits are just constant values.

```
1 # type suit = Hearts | Diamonds | Clubs | Spades;;  
2 type suit = Hearts | Diamonds | Clubs | Spades
```

Representing Cards

Cards do have associated data – suit and, for numbered cards, the number. So, this will be a disjoint union.

```
1 # type card = Ace of suit
2   | King of suit
3   | Queen of suit
4   | Jack of suit
5   | Num of suit * int;;
6 type card =
7   Ace of suit
8   | King of suit
9   | Queen of suit
10  | Jack of suit
11  | Num of suit * int
```

Printing Suits

Suits can be printed using a function and basic pattern matching.

```
1 #let name_of_suit s =  
2   match s with  
3     | Diamonds -> "Diamonds"  
4     | Hearts -> "Hearts"  
5     | Clubs -> "Clubs"  
6     | Spades -> "Spades";;  
7 val name_of_suit : suit -> string = <fun>
```

Printing Cards

Printing a description of the cards is a bit more involved, since it involves matching constructors and data values held with the constructors.

```
1 #let name_of_card c =  
2   match c with  
3     | Ace s -> "Ace of " ^ name_of_suit s  
4     | King s -> "King of " ^ name_of_suit s  
5     | Queen s -> "Queen of " ^ name_of_suit s  
6     | Jack s -> "Jack of " ^ name_of_suit s  
7     | Num (s,n) -> string_of_int n ^ " of " ^ name_of_suit s;;  
8 val name_of_card : card -> string = <fun>
```

Creating a Deck

We need to generate the deck of 52 cards to get started. We can do that using several helper functions.

```
1 let generate_deck =
2   let rec generate_list n m =
3     if n = m then [m] else n :: (generate_list (n + 1) m)
4   and generate_num_for_suit s =
5     List.map (fun n -> Num (s, n)) (generate_list 2 10)
6   and generate_suit s =
7     Ace s :: King s :: Queen s :: Jack s ::
8     generate_num_for_suit s
9   in
10  fun () -> List.fold_right (fun s y -> generate_suit s @ y)
11    [Hearts;Diamonds;Clubs;Spades] [];;
12 val generate_deck : unit -> card list = <fun>
```

The Created Deck

```
1 # generate_deck();  
2 - : card list =  
3 [Ace Hearts; King Hearts; Queen Hearts; Jack Hearts; Num (Hearts, 2);  
4  Num (Hearts, 3); Num (Hearts, 4); Num (Hearts, 5); Num (Hearts, 6);  
5  Num (Hearts, 7); Num (Hearts, 8); Num (Hearts, 9); Num (Hearts, 10);  
6  Ace Diamonds; King Diamonds; Queen Diamonds; Jack Diamonds;  
7  Num (Diamonds, 2); Num (Diamonds, 3); Num (Diamonds, 4); Num (Diamonds, 5);  
8  Num (Diamonds, 6); Num (Diamonds, 7); Num (Diamonds, 8); Num (Diamonds, 9);  
9  Num (Diamonds, 10); Ace Clubs; King Clubs; Queen Clubs; Jack Clubs;  
10 Num (Clubs, 2); Num (Clubs, 3); Num (Clubs, 4); Num (Clubs, 5);  
11 Num (Clubs, 6); Num (Clubs, 7); Num (Clubs, 8); Num (Clubs, 9);  
12 Num (Clubs, 10); Ace Spades; King Spades; Queen Spades; Jack Spades;  
13 Num (Spades, 2); Num (Spades, 3); Num (Spades, 4); Num (Spades, 5);  
14 Num (Spades, 6); Num (Spades, 7); Num (Spades, 8); Num (Spades, 9);  
15 Num (Spades, 10)]
```

Shuffling the Deck

To shuffle the deck, we somehow need to create a new deck from the old deck. Our strategy will be to randomly pick one card from the old deck and put it into the new deck until we are out of cards. First, we need a way to remove an element from a list.

Removing an Element from a List

To remove a single element from a list, we can write a function `all_but_nth`, which will give us back all but the `n`th element.

```
1 # let rec all_but_nth n l =  
2   match l with  
3     | [] -> []  
4     | x::xs ->  
5       match n with  
6         | 0 -> xs  
7         | _ -> x :: all_but_nth (n-1) xs;;  
8 val all_but_nth : int -> 'a list -> 'a list = <fun>
```

Shuffling the Deck

Now, we can randomly grab cards out of the old deck until we empty it. Note the use of `Random`, a provided module for random number generation.

```
1 # let shuffle_deck d =  
2   let () = Random.self_init()  
3   in let rec pick_until_done d =  
4       match d with  
5     | [] -> []  
6     | _ ->  
7       let n = Random.int (List.length d) in  
8         (List.nth d n) :: (pick_until_done (all_but_nth n d))  
9     in pick_until_done d;;  
10 val shuffle_deck : 'a list -> 'a list = <fun>
```

A Sample Shuffle

```
1 # let deck = shuffle_deck (generate_deck());  
2 val deck : card list =  
3   [Num (Clubs, 5); Jack Diamonds; Num (Spades, 5); Num (Clubs, 2);  
4     Num (Hearts, 4); Ace Clubs; Num (Clubs, 6); Num (Spades, 9); King Clubs;  
5     Queen Diamonds; Num (Spades, 3); Num (Hearts, 9); Ace Diamonds;  
6     Num (Clubs, 10); Num (Hearts, 6); Num (Hearts, 2); Num (Hearts, 8);  
7     Num (Clubs, 9); Queen Spades; Num (Spades, 2); Num (Hearts, 5);  
8     Queen Hearts; Num (Diamonds, 3); Ace Hearts; Num (Diamonds, 5);  
9     Num (Diamonds, 6); Num (Diamonds, 8); Num (Clubs, 4); Num (Diamonds, 10);  
10    King Diamonds; Num (Spades, 10); Num (Diamonds, 4); Num (Spades, 4);  
11    Jack Spades; Ace Spades; Num (Spades, 7); Num (Hearts, 3);  
12    Num (Diamonds, 7); Num (Hearts, 10); Jack Clubs; Num (Spades, 8);  
13    Queen Clubs; Num (Spades, 6); Num (Hearts, 7); Num (Clubs, 3);  
14    Num (Diamonds, 9); Num (Clubs, 8); King Hearts; Num (Diamonds, 2);  
15    Jack Hearts; Num (Clubs, 7); King Spades]
```

Dealing a Hand

Finally, with a shuffled deck, we can deal a hand of cards. In our case, we will always deal 5 cards, so we can just code that into the function.

```
1 # let deal d =  
2   match d with  
3     | (a::b::c::d::e::fs) -> [a;b;c;d;e],fs  
4     | _ -> [],d;;  
5 val deal : 'a list -> 'a list * 'a list = <fun>
```

A Sample Deal

```
1 # let (hand1,rest_of_deck) = deal deck;;
2 val hand1 : card list =
3   [Num (Clubs, 5); Jack Diamonds; Num (Spades, 5); Num (Clubs, 2);
4     Num (Hearts, 4)]
5 val rest_of_deck : card list =
6   [Ace Clubs; Num (Clubs, 6); Num (Spades, 9); King Clubs; Queen Diamonds;
7     Num (Spades, 3); Num (Hearts, 9); Ace Diamonds; Num (Clubs, 10);
8     Num (Hearts, 6); Num (Hearts, 2); Num (Hearts, 8); Num (Clubs, 9);
9     Queen Spades; Num (Spades, 2); Num (Hearts, 5); Queen Hearts;
10    Num (Diamonds, 3); Ace Hearts; Num (Diamonds, 5); Num (Diamonds, 6);
11    Num (Diamonds, 8); Num (Clubs, 4); Num (Diamonds, 10); King Diamonds;
12    Num (Spades, 10); Num (Diamonds, 4); Num (Spades, 4); Jack Spades;
13    Ace Spades; Num (Spades, 7); Num (Hearts, 3); Num (Diamonds, 7);
14    Num (Hearts, 10); Jack Clubs; Num (Spades, 8); Queen Clubs;
15    Num (Spades, 6); Num (Hearts, 7); Num (Clubs, 3); Num (Diamonds, 9);
16    Num (Clubs, 8); King Hearts; Num (Diamonds, 2); Jack Hearts;
17    Num (Clubs, 7); King Spades]
```

Printing the Hand

```
1 # let print_hand hand =
2   List.map (fun c -> name_of_card c) hand;;
3
4 # hand;;
5 - : card list =
6 [Queen Hearts; Num (Clubs, 9); Num (Hearts, 8);
7  Num (Hearts, 5); King Clubs]
8
9 # print_hand hand;;
10 - : string list =
11 ["Queen of Hearts"; "9 of Clubs"; "8 of Hearts";
12  "5 of Hearts"; "King of Clubs"]
```

Checking for Matching Rank

```
1 # let match_card_rank (c, c') =
2   match (c,c') with
3     | (Ace s, Ace s') -> true
4     | (King s, King s') -> true
5     | (Queen s, Queen s') -> true
6     | (Jack s, Jack s') -> true
7     | (Num (s, n), Num (s', n')) ->
8       if n = n' then true else false
9     | (_,_) -> false;;
10 val match_card_rank : card * card -> bool = <fun>
11
12 # match_card_rank (King Diamonds, King Clubs);;
13 - : bool = true
14 # match_card_rank (Num (Hearts,5), Num (Spades, 5));;
15 - : bool = true
16 # match_card_rank (King Clubs, Ace Hearts);;
17 - : bool = false
```