**Tutorial Exercise 1.**

The simulation above is very restricted in that it represents a hand simply as a pair of numbers. Suppose we also want to keep track of the actual cards in the hand, both their values and their suits. One way to do this is with a data abstraction `card` to represent a card, and, using `card`, to implement a data abstraction `card-set` to represent a set of cards. A `hand` in the above simulation would then be represented as a hand up-card together with a set of cards. Sketch a sample implementation that carries this out, using list structure to represent the cards and card sets. How do you need to change the procedures `make-hand`, `make-new-hand`, `hand-up-card`, `hand-total`, and `hand-add-card`? What else do you need to change in order to get the simulation to work (other than perhaps devising new strategies that take advantage of this extra information about hands)?

**Answer.**

Well, through data abstraction, a `card` can be represented by two numbers—the value of the card and a digit varies from 1 to 4 which stands for the suit of the card. Using this, we are now able to represent a `hand`, as is shown in Figure 1.

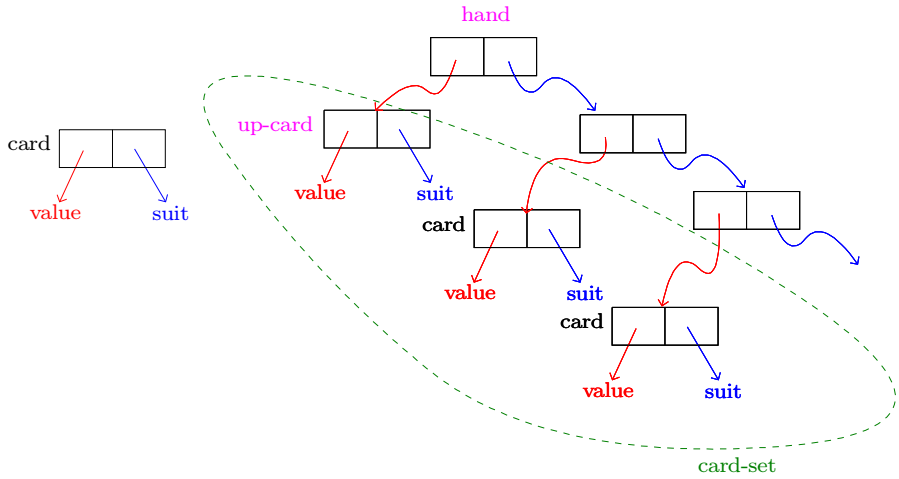


**Figure 1.** The Representation of a Card and a Hand

To implement a `card` in Lisp, we can use a constructor procedure `make-card` that creates a card from two kinds of data and two selector `card-value` and `card-suit`:

```
(define (make-card value suit)
  (cons value suit))

(define (card-value card)
  (car card))

(define (card-suit card)
  (cdr card))
```

Having these building blocks, we are now able to represent a `hand` inn a clear way:

```
(define (make-hand up-card card-set)
  (cons up-card card-set))

(define (hand-up-card hand)
  (car (car hand)))

(define (hand-total hand)
  (car (cdr hand)))
```