Rules for Templates

1. If the data has parts (e.g., a struct or a list), the template should pull out each of the parts using the appropriate selector (for structs), or by using first and rest (for lists).

Example:

(define-struct boa (name length food))

;; Boa is (make-boa String Natural String)

;; interp. A boa constrictor with a name, a length, and a favorite food

;; fcn-for-boa: Boa ->

;;

(define (fcn-for-boa a-boa)

 (boa-name a-boa) ;; String

 (boa-length a-boa) ;; Natural

 (boa-food a-boa)) ;; String

1. If the data can be one of several types (e.g., an itemization), the template should use a cond with a predicate for each type.

Example:

;; an Animal is one of

;; Boa

;; Dillo

;; Tiger

;; fcn-for-animal: Animal ->

;;

(define (fcn-for-animal an-ani)

 (cond [(boa? an-ani) …]

 [(dillo? an-ani) …]

 [(tiger? an-ani) …]))

Since each of the types in the cond questions shown above are types with parts, then each cond answer should extract those parts. Here we extract the parts by making a call to the appropriate template for the function that does the extraction:

;; fcn-for-animal: Animal ->

;;

(define (fcn-for-animal an-ani)

 (cond [(boa? an-ani) (fcn-for-boa an-ani)]

 [(dillo? an-ani) (fcn-for-dillo an-ani)]

 [(tiger? an-ani) (fcn-for-tiger an-ani)]))

1. If the data definition contains an arrow, the template should make a function call.

Example:

;; a ListOfNumber is one of

;; empty

;; (cons Number ListOfNumber)

;; fcn-for-lon: ListOfNumber ->

;;

(define (fcn-for-lon alon)

 (cond [(empty? alon) …]

 [(cons? alon)

(…(first alon)

 (fcn-for-lon (rest alon)))]))

The template shown above uses all three template rules. Make sure you can identify where each rule is being used.