Expanding the Zoo

We have snakes and armadillos. Let’s add ants.

An ant has

• a weight

• a location in the zoo

; An ant is
; (make-ant num posn)
(define-struct ant (weight loc))

(make-ant 0.001 (make-posn 4 5))
(make-ant 0.007 (make-posn 3 17))
Ants

(make-ant 0.001 (make-posn 4 5))

(make-ant 0.007 (make-posn 3 17))
Programming with Ants

Define `ant-at-home?`, which takes an ant and reports whether it is at the origin.
Contract, Purpose, and Header

; ant-at-home? : ant -> bool
Contract, Purpose, and Header

; ant-at-home? : ant -> bool
; Check whether ant a is home
Contract, Purpose, and Header

; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
  ...)

Examples

; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
    ...
)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0)))
    true)
(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1)))
    false)
Template

; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
  ...)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0))) true)
(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1))) false)
Template

; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
  ...)

; An ant is
; (make-ant num posn)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0)))
  true)
(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1)))
  false)
Template

; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
  ...
  (ant-weight a)
  ...
  (ant-loc a) ...
)

; An ant is
; (make-ant num posn)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0)))
  true)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1)))
  false)
Template

; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
  ...
  (ant-weight a)
  ...
  (posn-at-home? (ant-loc a)) ...
)

New template rule: data-defn reference ⇒ template reference

Add templates for referenced data, if needed, and implement body for referenced data

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0)))
  true)
(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1)))
  false)
; ant-at-home? : ant -> bool
; Check whether ant a is home
(define (ant-at-home? a)
  ...
  (ant-weight a)
  ...
  (posn-at-home? (ant-loc a)) ...)

(define (posn-at-home? p)
  ...
  (posn-x p) ...
  (posn-y p) ...
)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0)))
  true)

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1)))
  false)
Body

; ant-at-home? : ant -> bool
; Check whether ant a is home
;  (define (ant-at-home?  a)
;    ... (ant-weight a)
;    ... (posn-at-home? (ant-loc a)) ...)
; (define (posn-at-home?  p)
;   ... (posn-x p) ... (posn-y p) ...)
(define (ant-at-home?  a)
   (posn-at-home? (ant-loc a)))
(define (posn-at-home?  p)
   (and (= (posn-x p) 0) (= (posn-y p) 0)))

(check-expect (ant-at-home? (make-ant 0.001 (make-posn 0 0))) true)
(check-expect (ant-at-home? (make-ant 0.001 (make-posn 1 1))) false)
Shapes of Data and Templates

The shape of the template matches the shape of the data

; An ant is
; (make-ant num posn)

; A posn is
; (make-posn num num)

(define (ant-at-home? a)
  ... (ant-weight a)
  ... (posn-at-home? (ant-loc a)) ...)

(define (posn-at-home? p)
  ... (posn-x p) ... (posn-y p) ...)