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

\[
\text{(make-ant 0.001 (make-posn 4 5))}
\]

\[
\text{(make-ant 0.007 (make-posn 3 17))}
\]
Programming with Ants

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

Contract, Purpose, and Header
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)
  ... (ant-weight a)
  ... (ant-loc 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)
  ...
  (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)
; 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) ...
)
Programming with Ants

Define `feed-ant`, which feeds an ant 0.001 lbs of food

Define `move-ant`, which takes an ant, an amount to move X, and an amount to move Y, and returns a moved ant
Animals

All animals need to eat...

Define **feed-animal**, which takes an animal (snake, dillo, or ant) and feeds it (5 lbs, 2 lbs, or 0.001 lbs, respectively)

What is an **animal**?
Animal Data Definition

; An animal is either
;  - snake
;  - dillo
;  - ant

The “either” above makes this a new kind of data definition:

data with \textit{varieties}

Examples:

\begin{verbatim}
(make-snake 'slinky 10 'rats)
(make-dillo 2 true)
(make-ant 0.002 (make-posn 3 4))
\end{verbatim}
Feeding Animals

; feed-animal : animal -> animal
; To feed the animal a
(define (feed-animal a)
  ...)

(check-expect (feed-animal (make-snake 'Slinky 10 'rats))
  (make-snake 'Slinky 15 'rats))

(check-expect (feed-animal (make-dillo 2 true))
  (make-dillo 4 true))

(check-expect (feed-animal (make-ant 0.002 (make-posn 3 4)))
  (make-ant 0.003 (make-posn 3 4)))
Template for Animals

For the template step...

\[
\text{(define (feed-animal a) ...)}
\]

- Is `a` compound data?

- Technically yes, but the definition `animal` doesn’t have `make-something`, so we don’t use the compound-data template rule
Template for Varieties

Choice in the data definition

; An animal is either
; - snake
; - dillo
; - ant

means cond in the template:

(define (feed-animal a)
  (cond
    [.... ....]
    [.... ....]
    [.... ....]
    [.... ....]))

Three data choices means three cond cases
Questions for Varieties

\[
\text{(define} \ (\text{feed-}\text{animal} \ a) \\
\text{\hspace{1cm} (cond}} \\
\text{\hspace{2cm} [\ldots \ldots]} \\
\text{\hspace{2cm} [\ldots \ldots]} \\
\text{\hspace{2cm} [\ldots \ldots]} \\
\text{\hspace{2cm} [\ldots \ldots]]) \]
\]

How do we write a question for each case?

It turns out that
\[
\text{(define-}\text{struct} \ \text{snake} \ (\text{name} \ \text{weight} \ \text{food}))
\]
provides \text{snake}?

\[
\text{(snake?} \ (\text{make-}\text{snake} \ 'slinky \ 5 \ 'rats)) \rightarrow \text{true} \\
\text{(snake?} \ (\text{make-}\text{dillo} \ 2 \ \text{true})) \rightarrow \text{false} \\
\text{(snake?} \ 17) \rightarrow \text{false}
\]
(define (feed-animal a)
  (cond
    [(snake? a) ...]
    [(dillo? a) ...]
    [(ant? a) ...]))

New template rule: varieties → cond

Now continue template case-by-case...
Template

```
(define (feed-animal a)
  (cond
    [(snake? a) ... (feed-snake a) ...]
    [(dillo? a) ... (feed-dillo a) ...]
    [(ant? a) ... (feed-ant a) ...]]
)
```

Remember: references in the data definition ⇒ template references

; An animal is either
;  - snake
;  - dillo
;  - ant
Shapes of Data and Templates

; An animal is either
;  - snake
;  - dillo
;  - ant

; A snake is
; (make-snake sym num sym)

; A dillo is
; (make-dillo num bool)

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

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

(define (feed-animal a)
  (cond
   [(snake? a) ... (feed-snake a) ...]
   [(dillo? a) ... (feed-dillo a) ...]
   [(ant? a) ... (feed-ant a) ...]])

(define (feed-snake s)
  ... (snake-name s) ... (snake-weight s)
  ... (snake-food s) ...)

(define (feed-dillo d)
  ... (dillo-weight d)
  ... (dillo-alive? d) ...)

(define (feed-ant a)
  ... (ant-weight d)
  ... (feed-posn (ant-loc d)) ...)

(define (feed-posn p)
  ... (posn-x p) ... (posn-y p) ...)
### Design Recipe III

**Data**

- Understand the input data

**Contract, Purpose, and Header**

- Describe (but don’t write) the function

**Examples**

- Show what will happen when the function is done

**Template**

- Set up the body based on the input data (and *only* the input)

**Body**

- The most creative step: implement the function body

**Test**

- Run the examples
When the problem statement mentions $N$ different varieties of a thing, write a data definition of the form

```
; A thing is
;   - variety1
;   ...
;   - varietyN
```
Examples

When the input data has varieties, be sure to pick each variety at least once.

; An animal is either
; - snake
; - dillo
; - ant

(check-expect (feed-animal (make-snake 'Slinky 10 'rats))
  (make-snake 'Slinky 15 'rats))

(check-expect (feed-animal (make-dillo 2 true))
  (make-dillo 4 true))

(check-expect (feed-animal (make-ant 0.002 (make-posn 3 4)))
  (make-ant 0.003 (make-posn 3 4)))
When the input data has varieties, start with cond

- \( N \) varieties \( \Rightarrow N \) cond lines
- Formulate a question to match each corresponding variety
- Continue template steps case-by-case

```
(define (feed-animal a)
  (cond
    [(snake? a) ...]
    [(dillo? a) ...]
    [(ant? a) ...]))
```
When the input data has varieties, start with `cond`

- \(N\) varieties \(\Rightarrow\) \(N\) `cond` lines
- Formulate a question to match each corresponding variety
- Continue template steps case-by-case

When the data definition refers to a data definition, make the template refer to a template

```
(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) ...)
```
Template

When the input data has varieties, start with `cond`

- **N** varieties ⇒ **N** `cond` lines
- Formulate a question to match each corresponding variety
- Continue template steps case-by-case

When the data definition refers to a data definition, make the template refer to a template

```
(define (feed-animal a)
  (cond
    [(snake? a) ... (feed-snake a) ...]
    [(dillo? a) ... (feed-dillo a) ...]
    [(ant? a) ... (feed-ant a) ...]))
```