

**Part III:
Compound
Data of
Arbitrary
Size**

**Marco T.
Morazán**

Lists

**List
Processing**

**Natural
Numbers**

**Interval
Processing**

**Aliens Attack
Version 4**

Binary Trees

**Mutually
Recursive
Data**

**Processing
Multiple
Inputs of
Arbitrary
Size**

Part III: Compound Data of Arbitrary Size

Marco T. Morazán

Seton Hall University

Outline

1 Lists

2 List Processing

3 Natural Numbers

4 Interval Processing

5 Aliens Attack Version 4

6 Binary Trees

7 Mutually Recursive Data

8 Processing Multiple Inputs of Arbitrary Size

Part III: Compound Data of Arbitrary Size

Marco T.
Morazán

Lists

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- Congratulations! Change your language Intermediate Student with lambda (ISL+)
- BSL is a subset of ISL+

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- We begin the study of data of arbitrary size
- Data whose size is not constant
- Not every instance of the data shall be of the same size

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- We begin the study of data of arbitrary size
- Data whose size is not constant
- Not every instance of the data shall be of the same size
- A grocery lists contain a different number of items
- A grocery list is data of arbitrary size.

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- We begin the study of data of arbitrary size
- Data whose size is not constant
- Not every instance of the data shall be of the same size
- A grocery lists contain a different number of items
- A grocery list is data of arbitrary size.
- Lists may be used to represent much more than just the groceries
- In Aliens Attack we would like to have more than one alien and one shot
- Is the number of aliens constant? Is the number of shots constant?

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- We begin the study of data of arbitrary size
- Data whose size is not constant
- Not every instance of the data shall be of the same size
- A grocery lists contain a different number of items
- A grocery list is data of arbitrary size.
- Lists may be used to represent much more than just the groceries
- In Aliens Attack we would like to have more than one alien and one shot
- Is the number of aliens constant? Is the number of shots constant?
- As the game advances the number of aliens is not constant
- The number of shots varies

Lists

Creating and Accessing Lists in ISL+

- The smallest possible list is the empty list
- In ISL+: '()

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Creating and Accessing Lists in ISL+

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- The smallest possible list is the empty list
- In ISL+: '()
- We can use the empty list to define many different types of list
- An empty grocery list, an empty list of aliens, and an empty list of shots:

```
(define E-GLIST '())
(define E-LOA   '())
(define E-LOS   '())

(check-expect (empty? E-GLIST) #true)
(check-expect (empty? E-LOA)    #true)
(check-expect (empty? E-LOS)    #true)
```

- '() is an ISL+ constant that represents any type of empty list
- The ISL+ predicate empty? returns true if its input is '() and #false otherwise

Lists

Creating and Accessing Lists in ISL+

- Non-empty lists may be constructed by adding an element to the front of an existing list
- Constructor: `cons`

`any list → list`

`Purpose:` To construct a new list by adding the given value to the front of the given list

Lists

Creating and Accessing Lists in ISL+

- Non-empty lists may be constructed by adding an element to the front of an existing list
- Constructor: `cons`

any list → list

Purpose: To construct a new list by adding the given value to the front of the given list

- We can now use `cons` to build a grocery list, a list of aliens, and a list of shots of any (finite) size:

```
(define A-GLIST (cons "milk"
                        (cons "apples"
                              E-GLIST)))
(define A-LOA   (cons (make-posn 10 2)
                      (cons (make-posn 5 12)
                            (cons (make-posn 4 8)
                                  (cons (make-posn 15 7)
                                        (cons (make-posn 6 6)
                                              E-LOA))))))
(define A-LOS   (cons (make-posn 4 2)
                      (cons NO-SHOT
                            (cons (make-posn 17 3)
                                  E-LOS))))
(check-expect (cons? E-GLIST) #false)
(check-expect (cons? E-LOA) #false)
(check-expect (cons? E-LOS) #false)
(check-expect (cons? A-GLIST) #true)
(check-expect (cons? A-LOA) #true)
(check-expect (cons? A-LOS) #true)
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Creating and Accessing Lists in ISL+

- Selectors are needed to extract the elements in a list
- Two selectors: `first` and `rest`

Lists

Creating and Accessing Lists in ISL+

- Selectors are needed to extract the elements in a list
- Two selectors: `first` and `rest`
- Care must be taken when using these selector functions because the empty list does not have a first element nor does it have a value for the rest of the elements:

```
(check-error (first E-GLIST))
(check-error (first E-LOA))
(check-error (first E-LOS))
(check-expect (first A-GLIST) "milk")
(check-expect (first A-LOA) (make-posn 10 2))
(check-expect (first A-LOS) (make-posn 4 2))
(check-error (rest E-GLIST))
(check-error (rest E-LOA))
(check-error (rest E-LOS))
(check-expect (rest A-GLIST) (cons "apples" E-GLIST))
(check-expect (rest A-LOA)
              (cons (make-posn 5 12)
                    (cons (make-posn 4 8)
                          (cons (make-posn 15 7)
                                (cons (make-posn 6 6) E-LOA))))))
(check-expect (rest A-LOS)
              (cons NO-SHOT (cons (make-posn 17 3) E-LOS)))
```

- Special form of `check-error`: does not require an error message to check

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Creating and Accessing Lists in ISL+

- How is the second element of a list extracted? The third element?

Lists

Creating and Accessing Lists in ISL+

- How is the second element of a list extracted? The third element?
- To achieve extractions beyond the first list element you need to compose calls to `first` and `rest`:

```
(first (rest A-LOA)) = (first
                           (rest (cons (make-posn 4 2)
                                       (cons NO-SHOT
                                             (cons
                                               (make-posn 17 3)
                                               E-LOS))))
                           = (first (cons NO-SHOT
                                         (cons (make-posn 17 3)
                                               E-LOS)))
                           = NO-SHOT
```

Lists

List Processing

Natural Numbers

Interval Processing

Aliens Attack Version 4

Binary Trees

Mutually Recursive Data

Processing Multiple Inputs of Arbitrary Size

Lists

Creating and Accessing Lists in ISL+

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- How is the second element of a list extracted? The third element?
- To achieve extractions beyond the first list element you need to compose calls to `first` and `rest`:

```
(first (rest A-LOA)) = (first
                           (rest (cons (make-posn 4 2)
                                       (cons NO-SHOT
                                             (cons
                                               (make-posn 17 3)
                                               E-LOS))))
                           = (first (cons NO-SHOT
                                         (cons (make-posn 17 3)
                                               E-LOS)))
                           = NO-SHOT
```

- Extracting up to the eighth element: `second`, `third`, `fourth`, and so on up to `eighth`
- With a list that is too short an error is thrown

Lists

Shorthand for Building Lists

- Consider constructing a list of first 7 digits:

```
(define SEVEN-DIGITS
  (cons
    0
    (cons 1
      (cons 2
        (cons 3
          (cons 4
            (cons 5
              (cons 6 '()))))))))
```

```
(check-expect (first SEVEN-DIGITS) 0)
(check-expect (second SEVEN-DIGITS) 1)
(check-expect (third SEVEN-DIGITS) 2)
(check-expect (fourth SEVEN-DIGITS) 3)
(check-expect (fifth SEVEN-DIGITS) 4)
(check-expect (sixth SEVEN-DIGITS) 5)
(check-expect (seventh SEVEN-DIGITS) 6)
```

- A lot of repetition

Lists

Shorthand for Building Lists

- Consider constructing a list of first 7 digits:

```
(define SEVEN-DIGITS
  (cons
    0
    (cons 1
      (cons 2
        (cons 3
          (cons 4
            (cons 5
              (cons 6 '()))))))))
```

```
(check-expect (first SEVEN-DIGITS) 0)
(check-expect (second SEVEN-DIGITS) 1)
(check-expect (third SEVEN-DIGITS) 2)
(check-expect (fourth SEVEN-DIGITS) 3)
(check-expect (fifth SEVEN-DIGITS) 4)
(check-expect (sixth SEVEN-DIGITS) 5)
(check-expect (seventh SEVEN-DIGITS) 6)
```

- A lot of repetition
- What if you are now asked to define a list with the integers in [0..19]?
- Write cons twenty times?

Lists

Shorthand for Building Lists

- Consider constructing a list of first 7 digits:

```
(define SEVEN-DIGITS
  (cons
    0
    (cons 1
      (cons 2
        (cons 3
          (cons 4
            (cons 5
              (cons 6 '())))))))))))

(check-expect (first SEVEN-DIGITS) 0)
(check-expect (second SEVEN-DIGITS) 1)
(check-expect (third SEVEN-DIGITS) 2)
(check-expect (fourth SEVEN-DIGITS) 3)
(check-expect (fifth SEVEN-DIGITS) 4)
(check-expect (sixth SEVEN-DIGITS) 5)
(check-expect (seventh SEVEN-DIGITS) 6)
```

- A lot of repetition
- What if you are now asked to define a list with the integers in [0..19]?
- Write cons twenty times?
- To avoid all this repetition ISL+ provides 3 shorthand constructors for lists

Lists

Shorthand for Building Lists

- The first is used when the elements of the list are known in advance

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Shorthand for Building Lists

- The first is used when the elements of the list are known in advance
- A *quoted list* has the elements listed inside parenthesis preceded by a ':

```
(define SEVEN-DIGITS '(0 1 2 3 4 5 6))
```

```
(check-expect (first SEVEN-DIGITS) 0)
(check-expect (second SEVEN-DIGITS) 1)
(check-expect (third SEVEN-DIGITS) 2)
(check-expect (fourth SEVEN-DIGITS) 3)
(check-expect (fifth SEVEN-DIGITS) 4)
(check-expect (sixth SEVEN-DIGITS) 5)
(check-expect (seventh SEVEN-DIGITS) 6)
```

Lists

Shorthand for Building Lists

- The first is used when the elements of the list are known in advance
- A *quoted list* has the elements listed inside parenthesis preceded by a ':

```
(define SEVEN-DIGITS '(0 1 2 3 4 5 6))
```

```
(check-expect (first SEVEN-DIGITS) 0)
(check-expect (second SEVEN-DIGITS) 1)
(check-expect (third SEVEN-DIGITS) 2)
(check-expect (fourth SEVEN-DIGITS) 3)
(check-expect (fifth SEVEN-DIGITS) 4)
(check-expect (sixth SEVEN-DIGITS) 5)
(check-expect (seventh SEVEN-DIGITS) 6)
```

- Nothing after the quote (inside the parenthesis) is evaluated
-

```
(define SOME-SQRS1 (cons (sqr 0)
                           (cons (sqr 1)
                                 (cons (sqr 2)
                                       empty))))
```

```
(define SOME-SQRS2 '((sqr 0) (sqr 1) (sqr 2)))
```

```
(check-expect SOME-SQRS1 '(0 1 4))
(check-expect (not (equal? SOME-SQRS1 SOME-SQRS2)) #true)
```

Lists

Shorthand for Building Lists

- The second allows you to create a list from evaluated expressions
- The shorthand provided by ISL+: list

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Shorthand for Building Lists

- The second allows you to create a list from evaluated expressions
- The shorthand provided by ISL+: list
- Evaluates all of its arguments and creates a list of the results.

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Shorthand for Building Lists

- The second allows you to create a list from evaluated expressions
- The shorthand provided by ISL+: list
- Evaluates all of its arguments and creates a list of the results.
- Refactor the SOME-SQRS2 definition and update the tests as follows:

```
(define SOME-SQRS1 (cons (sqr 0)
                           (cons (sqr 1)
                                  (cons (sqr 2)
                                        '()))))
```

```
(define SOME-SQRS2 (list (sqr 0) (sqr 1) (sqr 2)))

(check-expect SOME-SQRS1 '(0 1 4))
(check-expect (equal? SOME-SQRS1 SOME-SQRS2) #true)
```

- Use list when you want all the expressions to be evaluated to construct a list.

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Shorthand for Building Lists

- The third shorthand is a *quasiquoted list*
- A quasiquoted list is a combination of ' and list

Lists

Shorthand for Building Lists

- The third shorthand is a *quasiquoted list*
- A quasiquoted list is a combination of ' and list
- Instead of preceding the opening parentheses with a ', it is preceded with `
- The ` indicates that some subexpression inside the parenthesis may have to be evaluated
- To indicate that an expression needs to be evaluated it must be preceded by a ,

Lists

Shorthand for Building Lists

- The third shorthand is a *quasiquoted list*
- A quasiquoted list is a combination of ' and list
- Instead of preceding the opening parentheses with a ', it is preceded with `
- The ` indicates that some subexpression inside the parenthesis may have to be evaluated
- To indicate that an expression needs to be evaluated it must be preceded by a ,
 - (define X 2) (define Y 3) (define Z 4)
 - (define A-LIST `((add1 X) J (* Z Y)))
 - (define A-LIST2 `,,(add1 X) J ,(,* Z Y)))
 - (check-expect (first A-LIST) '(add1 X))
 - (check-expect (first A-LIST2) 3)
 - (check-expect (second A-LIST) 'J)
 - (check-expect (second A-LIST2) 'J)
 - (check-expect (third A-LIST) '(* Z Y))
 - (check-expect (third A-LIST2) 12)
- (add1 X) and (* Z Y) are evaluated to construct A-LIST2
- Treated as literal values for A-LIST because they are inside a quoted list.

Lists

Recursive Data Definitions

- How is a list processed?

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Recursive Data Definitions

- How is a list processed?
- We need a data definition and a function template

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

Recursive Data Definitions

- How is a list processed?
- We need a data definition and a function template
- Consider defining a list of numbers:

```
'()  
(cons 87 '())  
(cons 24 (cons 87 '()))  
(cons 16 (cons 24 (cons 87 '()))))  
(cons 31 (cons 16 (cons 24 (cons 87 '())))))
```

Lists

Recursive Data Definitions

- How is a list processed?
- We need a data definition and a function template
- Consider defining a list of numbers:

```
'()  
(cons 87 '())  
(cons 24 (cons 87 '()))  
(cons 16 (cons 24 (cons 87 '()))))  
(cons 31 (cons 16 (cons 24 (cons 87 '())))))
```

- There are two list-of-numbers varieties
- The empty list of numbers
- The non-empty list of numbers built using cons
- Data definition:

```
;; A list of numbers (lon) is either  
;; 1. '()  
;; 2. (cons ??? ???)
```

Lists

Recursive Data Definitions

- How is a list processed?
- We need a data definition and a function template
- Consider defining a list of numbers:

```
'()  
(cons 87 '())  
(cons 24 (cons 87 '()))  
(cons 16 (cons 24 (cons 87 '()))))  
(cons 31 (cons 16 (cons 24 (cons 87 '())))))
```

- There are two list-of-numbers varieties
- The empty list of numbers
- The non-empty list of numbers built using `cons`
- Data definition:

```
;; A list of numbers (lon) is either  
;; 1. '()  
;; 2. (cons ??? ???)
```

- The first argument must be a number:

```
;; A list of numbers (lon) is either  
;; 1. '()  
;; 2. (cons number ???)
```

- What is the type of the second argument to `cons`?

Lists

Recursive Data Definitions

- The type of the second argument to `cons` is lon:

```
;; A list of number (lon) is either
;; 1. '()
;; 2. (cons number lon)
```

- A list of numbers is defined in terms of itself (*circular*)
- A type defined in terms of itself is a *recursive* data type
- Does this make any sense? Can this data definition be used to build any list of numbers?

Lists

Recursive Data Definitions

- The type of the second argument to `cons` is `lon`:

```
;; A list of number (lon) is either
;; 1. '()
;; 2. (cons number lon)
```

- A list of numbers is defined in terms of itself (*circular*)
- A type defined in terms of itself is a *recursive* data type
- Does this make any sense? Can this data definition be used to build any list of numbers?
- Build '`(cons 24 (cons 87 '()))`:

```
lon → (cons 24 lon)           using rule 2 to substitute lon
      → (cons 24
            (cons 87 lon))   using rule 2 to substitute lon
      → (cons 24
            (cons 87 '())) using rule 1 to substitute lon
```

- This is an example of a *derivation*
- A derivation is the series of steps used to show how to build an instance of a data type using the varieties of a data definition.

Lists

Recursive Data Definitions

- The type of the second argument to `cons` is `lon`:

```
;; A list of number (lon) is either
;; 1. '()
;; 2. (cons number lon)
```

- A list of numbers is defined in terms of itself (*circular*)
- A type defined in terms of itself is a *recursive* data type
- Does this make any sense? Can this data definition be used to build any list of numbers?
- Build '`(cons 24 (cons 87 '()))`:

```
lon → (cons 24 lon)           using rule 2 to substitute lon
      → (cons 24
            (cons 87 lon))   using rule 2 to substitute lon
      → (cons 24
            (cons 87 '())) using rule 1 to substitute lon
```

- This is an example of a *derivation*
- A derivation is the series of steps used to show how to build an instance of a data type using the varieties of a data definition.
- Derive '`(cons 42 (cons -6 (cons "Hi" (cons 87 '()))))`:

```
lon → (cons 42 lon)  using rule 2 to substitute lon
      → (cons 42
            (cons -6 lon)) using rule 2 to substitute lon
```

Lists

Recursive Data Definitions

- The type of the second argument to `cons` is `lon`:

```
;; A list of number (lon) is either
;; 1. '()
;; 2. (cons number lon)
```

- A list of numbers is defined in terms of itself (*circular*)
- A type defined in terms of itself is a *recursive* data type
- Does this make any sense? Can this data definition be used to build any list of numbers?
- Build '`(cons 24 (cons 87 '()))`:

```
lon → (cons 24 lon)           using rule 2 to substitute lon
      → (cons 24
            (cons 87 lon))   using rule 2 to substitute lon
      → (cons 24
            (cons 87 '())) using rule 1 to substitute lon
```

- This is an example of a *derivation*
 - A derivation is the series of steps used to show how to build an instance of a data type using the varieties of a data definition.
 - Derive '`(cons 42 (cons -6 (cons "Hi" (cons 87 '()))))`:
- ```
lon → (cons 42 lon) using rule 2 to substitute lon
 → (cons 42
 (cons -6 lon)) using rule 2 to substitute lon
```
- Fails: no `lon` variety whose first element is a string **Typo in textbook**
  - '`(cons 42 (cons -6 (cons "Hi" (cons 87 '()))))` is not a `lon`

# Lists

## Recursive Data Definitions

- It is very likely that you have been taught to steer away from recursive data definitions
- This is unfortunate because as we have seen recursive data definitions are useful to define data of arbitrary size like lists
- Any data of arbitrary size requires a recursive data definition
- Why are students steered away from recursive data definitions?

# Lists

## Recursive Data Definitions

- It is very likely that you have been taught to steer away from recursive data definitions
- This is unfortunate because as we have seen recursive data definitions are useful to define data of arbitrary size like lists
- Any data of arbitrary size requires a recursive data definition
- Why are students steered away from recursive data definitions?
- Likely rooted in the fact that recursive data definitions may be nonsense:  
`; A list of numbers (lon2) is a (cons number lon2)`
- Is this a useful data definition?

# Lists

## Recursive Data Definitions

- It is very likely that you have been taught to steer away from recursive data definitions
- This is unfortunate because as we have seen recursive data definitions are useful to define data of arbitrary size like lists
- Any data of arbitrary size requires a recursive data definition
- Why are students steered away from recursive data definitions?
- Likely rooted in the fact that recursive data definitions may be nonsense:  

```
; A list of numbers (lon2) is a (cons number lon2)
```
- Is this a useful data definition?
- Try to derive '(cons 24 (cons 87 '())):  

```
lon2 → (cons 24 lon2) substitute lon2 using rule 2
 → (cons 24
 (cons 87 lon2)) substitute lon2 using rule 2
```
- The derivation fails because we are unable to instantiate the lon2 in (cons 87 lon2)

# Lists

## Recursive Data Definitions

- It is very likely that you have been taught to steer away from recursive data definitions
- This is unfortunate because as we have seen recursive data definitions are useful to define data of arbitrary size like lists
- Any data of arbitrary size requires a recursive data definition
- Why are students steered away from recursive data definitions?
- Likely rooted in the fact that recursive data definitions may be nonsense:  
$$\text{; A list of numbers (lon2) is a (cons number lon2)}$$
- Is this a useful data definition?
- Try to derive '(cons 24 (cons 87 '())):  
$$\begin{aligned} \text{lon2} &\rightarrow (\text{cons } 24 \text{ lon2}) && \text{substitute lon2 using rule 2} \\ &\rightarrow (\text{cons } 24 && \\ &&& (\text{cons } 87 \text{ lon2})) && \text{substitute lon2 using rule 2} \end{aligned}$$
- The derivation fails because we are unable to instantiate the lon2 in (cons 87 lon2)
- You may say that clearly it should be '()
- In fact, this would be wrong because this data definition does it say that '() is a lon2

# Lists

## Recursive Data Definitions

- This leads to asking ourselves what constitutes a useful recursive data definition

# Lists

## Recursive Data Definitions

- This leads to asking ourselves what constitutes a useful recursive data definition
- In order to be useful a recursive data definition must have the following characteristics:
  - ① At least two subtypes (varieties)
  - ② At least one subtype that does not contain a selfreference
  - ③ At least one subtype that contains a selfreference
- The subtypes that do not contain a selfreference are known as *base subtypes*.
- The subtypes that do contain a selfreference are known as *recursive subtypes*

# Lists

## Recursive Data Definitions

- This leads to asking ourselves what constitutes a useful recursive data definition
- In order to be useful a recursive data definition must have the following characteristics:
  - ① At least two subtypes (varieties)
  - ② At least one subtype that does not contain a selfreference
  - ③ At least one subtype that contains a selfreference
- The subtypes that do not contain a selfreference are known as *base subtypes*.
- The subtypes that do contain a selfreference are known as *recursive subtypes*
- In the recursive data definition for lon, we have '()' as a base subtype and (cons number lon) as a recursive subtype
- The recursive data definition for lon2 is not useful because it does not have a base subtype.

# Lists

## Recursive Data Definitions

- We can now define a representation for multiple aliens and multiple shots in Aliens Attack

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Lists

## Recursive Data Definitions

- We can now define a representation for multiple aliens and multiple shots in Aliens Attack
- Both are data of arbitrary size: we need a recursive data definitions

# Lists

## Recursive Data Definitions

- We can now define a representation for multiple aliens and multiple shots in Aliens Attack

- Both are data of arbitrary size: we need a recursive data definitions
- `;; A list of alien (loa) is either:`

```
;; 1. '()
;; 2. (cons alien loa)
```

`; Sample instances of loa`

```
(define E-LOA '())
(define INIT-LOA
 (list
 (make-posn 8 0) (make-posn 9 0) (make-posn 10 0)
 (make-posn 11 0) (make-posn 12 0) (make-posn 13 0)
 (make-posn 8 1) (make-posn 9 1) (make-posn 10 1)
 (make-posn 11 1) (make-posn 12 1) (make-posn 13 1)
 (make-posn 8 2) (make-posn 9 2) (make-posn 10 2)
 (make-posn 11 2) (make-posn 12 2) (make-posn 13 2)))
```

# Lists

## Recursive Data Definitions

- We can now define a representation for multiple aliens and multiple shots in Aliens Attack

- Both are data of arbitrary size: we need a recursive data definitions
- `;; A list of alien (loa) is either:`

```
;; 1. '()
;; 2. (cons alien loa)
```

`; Sample instances of loa`

```
(define E-LOA '())
(define INIT-LOA
 (list
```

```
 (make-posn 8 0) (make-posn 9 0) (make-posn 10 0)
 (make-posn 11 0) (make-posn 12 0) (make-posn 13 0)
 (make-posn 8 1) (make-posn 9 1) (make-posn 10 1)
 (make-posn 11 1) (make-posn 12 1) (make-posn 13 1)
 (make-posn 8 2) (make-posn 9 2) (make-posn 10 2)
 (make-posn 11 2) (make-posn 12 2) (make-posn 13 2)))
```

- `;; A list of shot (los) is either:`

```
;; 1. '()
;; 2. (cons shot los)
```

`; Sample instances of los`

```
(define INIT-LOS '())
```

```
(define LOS2 (list (make-posn 8 0) (make-posn 10 5)))
```

## Part III: Compound Data of Arbitrary Size

Marco T.  
Morazán

### Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

## Lists Homework

- Problems: 117–120

# Lists

## Generic Data Definitions

- A los is either:
  1. ()
  2. (cons shot los)A loa is either:
  1. ()
  2. (cons alien loa)A lon is either:
  1. ()
  2. (cons number lon)
- Almost identical
- Can we apply an abstraction step to avoid repetitions among data definitions?

# Lists

## Generic Data Definitions

- A los is either:
  1. '()
  2. (cons shot los)A loa is either:
  1. '()
  2. (cons alien loa)A lon is either:
  1. '()
  2. (cons number lon)
- Almost identical
- Can we apply an abstraction step to avoid repetitions among data definitions?
- The difference is a type
- Capture the differences using type variables

# Lists

## Generic Data Definitions

- A los is either:
  1. '()
  2. (cons shot los)A loa is either:
  1. '()
  2. (cons alien loa)A lon is either:
  1. '()
  2. (cons number lon)
- Almost identical
- Can we apply an abstraction step to avoid repetitions among data definitions?
- The difference is a type
- Capture the differences using type variables
- Let X be the difference:

```
;; A list of X ((listof X)) is either:
;; 1. '()
;; 2. (cons X (listof X))
```

# Lists

## Generic Data Definitions

- A los is either:
  1. '()
  2. (cons shot los)A loa is either:
  1. '()
  2. (cons alien loa)A lon is either:
  1. '()
  2. (cons number lon)

- Almost identical
- Can we apply an abstraction step to avoid repetitions among data definitions?
- The difference is a type
- Capture the differences using type variables
- Let X be the difference:

```
;; A list of X ((listof X)) is either:
;; 1. '()
;; 2. (cons X (listof X))
```

- This data definition works for many different types
- A data definition that works for many different types is called a **generic** (or **parameterized**) data definition
- We may use (listof X) in our signatures

# Lists

## Function Templates for Lists

- `;; A list of X ((listof X)) is either:  
;; 1. '()  
;; 2. (cons X (listof X))`

# Lists

## Function Templates for Lists

- ```
;; A list of X ((listof X)) is either:  
;; 1. '()  
;; 2. (cons X (listof X))
```
- ```
;; Sample instances of (listof X)
(define LOX1 ...)
(define LOX2 ...) ...
```

# Lists

## Function Templates for Lists

- ```
;; A list of X ((listof X)) is either:  
;; 1. '()  
;; 2. (cons X (listof X))
```
- ```
;; Sample instances of (listof X)
(define LOX1 ...)
(define LOX2 ...) ...
```

- ```
;; Sample expressions for f-on-loX  
(define LOX1-VAL ...)  
(define LOX2-VAL ...) ...
```

Lists

Function Templates for Lists

- ```
;; A list of X ((listof X)) is either:
;; 1. '()
;; 2. (cons X (listof X))
```
- ```
;; Sample instances of (listof X)  
(define LOX1 ...)  
(define LOX2 ...) ...
```
- ```
;; (listof X) ... → ...
;; Purpose: ...
(define (f-on-loX a-loX ...))
```

- ```
;; Sample expressions for f-on-loX  
(define LOX1-VAL ...)  
(define LOX2-VAL ...) ...
```

Lists

Function Templates for Lists

- ```
;; A list of X ((listof X)) is either:
;; 1. '()
;; 2. (cons X (listof X))
```
- ```
;; Sample instances of (listof X)  
(define LOX1 ...)  
(define LOX2 ...) ...
```
- ```
;; (listof X) ... → ...
;; Purpose: ...
(define (f-on-loX a-loX ...))
```

- ```
;; Sample expressions for f-on-loX  
(define LOX1-VAL ...)  
(define LOX2-VAL ...) ...
```
 - ```
;; Tests using sample computations for f-on-loX
(check-expect (f-on-loX LOX1 ...) LOX1-VAL)
(check-expect (f-on-loX LOX2 ...) LOX2-VAL) ...
```
- 
- ```
;; Tests using sample values for f-on-loX  
(check-expect (f-on-loX ...) ...) ...
```

Lists

Function Templates for Lists

- ```
;; A list of X ((listof X)) is either:
;; 1. ()
;; 2. (cons X (listof X))
```
  - ```
;; Sample instances of (listof X)  
(define LOX1 ...)  
(define LOX2 ...) ...
```
 - ```
;; (listof X) ... → ...
;; Purpose: ...
(define (f-on-loX a-loX ...)
```
  - ```
(if (empty? a-loX)  
    ...  
    ...)
```
 - ```
;; Sample expressions for f-on-loX
(define LOX1-VAL ...)
(define LOX2-VAL ...) ...
```
  - ```
;; Tests using sample computations for f-on-loX  
(check-expect (f-on-loX LOX1 ...) LOX1-VAL)  
(check-expect (f-on-loX LOX2 ...) LOX2-VAL) ...
```
- ```
;; Tests using sample values for f-on-loX
(check-expect (f-on-loX ...) ...)
```

# Lists

## Function Templates for Lists

- ```
•     ;; A list of X ((listof X)) is either:  
•       ;; 1. ()  
•       ;; 2. (cons X (listof X))  
•     ;; Sample instances of (listof X)  
(define LOX1 ...)  
(define LOX2 ...) ...  
•     ;; (listof X) ... → ...  
•     ;; Purpose: ...  
(define (f-on-loX a-loX ...)  
•       (if (empty? a-loX)  
•           ...  
•           ...  
•             (f-on-X (first a-loX))...  
•             (f-on-loX (rest a-loX) ...)...))  
•     ;; Sample expressions for f-on-loX  
(define LOX1-VAL ...)  
(define LOX2-VAL ...) ...  
•     ;; Tests using sample computations for f-on-loX  
(check-expect (f-on-loX LOX1 ...) LOX1-VAL)  
(check-expect (f-on-loX LOX2 ...) LOX2-VAL) ...  
•     ;; Tests using sample values for f-on-loX  
(check-expect (f-on-loX ...) ...)
```

Lists

Function Templates for Lists

- ```
;; (listof X) ... → ...
;; Purpose: ...
(define (f-on-loX a-loX ...)
 (if (empty? a-loX)
 ...
 ... (f-on-X (first a-loX))... (f-on-loX (rest a-loX) ...)...))
```
- Every selfreference in a data definition becomes a selfreference in the definition template and, eventually, a selfreference in a function definition
- *Data of arbitrary size is processed by a recursive function*

# Lists

## Function Templates for Lists

- ```
;; (listof X) ... → ...
;; Purpose: ...
(define (f-on-loX a-loX ...)
  (if (empty? a-loX)
      ...
      ... (f-on-X (first a-loX))... (f-on-loX (rest a-loX) ...)...))
```
- Every selfreference in a data definition becomes a selfreference in the definition template and, eventually, a selfreference in a function definition
- *Data of arbitrary size is processed by a recursive function*
- Recursion that is based on the structure of your data is called *structural recursion* (or natural recursion)

Lists

Function Templates for Lists

- ```
;; (listof X) ... → ...
;; Purpose: ...
(define (f-on-loX a-loX ...)
 (if (empty? a-loX)
 ...
 ... (f-on-X (first a-loX))... (f-on-loX (rest a-loX) ...)...))
```
- Every selfreference in a data definition becomes a selfreference in the definition template and, eventually, a selfreference in a function definition
- *Data of arbitrary size is processed by a recursive function*
- Recursion that is based on the structure of your data is called *structural recursion* (or natural recursion)
- Divide-and-conquer strategy naturally arises
- A subproblem for an X and a subproblem for a (smaller) (listof X) must be solved
- It is always the case that the subproblem is smaller and, therefore, closer to a base case
- This means that when structural recursion is correctly used your code is guaranteed to eventually stop

# Lists

## Designing List-Processing Functions

- We can use the generic data definition for a (listof X) to define a list of numbers:

A lon is a (listof number)

# Lists

## Designing List-Processing Functions

- We can use the generic data definition for a (listof X) to define a list of numbers:

A lon is a (listof number)

- In other words:

```
;; A list of number (lon) is either
;; 1. '()
;; 2. (cons number lon)
```

# Lists

## Designing List-Processing Functions

- We can use the generic data definition for a (listof X) to define a list of numbers:

A lon is a (listof number)

- In other words:

```
;; A list of number (lon) is either
;; 1. '()
;; 2. (cons number lon)
```

- Function template for a lon:

```
;; Sample instances of lon
(define LON1 ...) (define LON2 ...) ...
;; lon ... → ... Purpose: ...
(define (f-on-lon a-lon ...)
 (if (empty? a-lon)
 ...
 ... (f-on-number (first a-lon)) ... (f-on-lon (rest a-lon)))
 ; Sample expressions for f-on-lon
 (define LON1-VAL ...)
 (define LON2-VAL ...) ...
 ; Tests using sample computations for f-on-lon
 (check-expect (f-on-lon LON1 ...) LON1-VAL)
 (check-expect (f-on-lon LON2 ...) LON2-VAL) ...
 ; Tests using sample values for f-on-lon
 (check-expect (f-on-lon ...) ...))
```

# Lists

## Designing List-Processing Functions

- Write a function that squares a list of numbers

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Lists

## Designing List-Processing Functions

- Write a function that squares a list of numbers

- ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```

Lists

Designing List-Processing Functions

- Write a function that squares a list of numbers

- :: Sample instances of long

```
(define LON1 '())
```

```
(define LON3 '(1 3 3 4))
```

- 1

:: Sample expressions for square-long

```
(define LON1-VAL '())
```

```
(define LON2-VAL (cons (sqr (first '(1 2 3 4))))
```

```
(square-lon (rest '(1 2 3 4))))
```

Lists

Designing List-Processing Functions

- Write a function that squares a list of numbers

- :: Sample instances of long

```
(define LON1 '())
```

```
(define L0N2 '(1 2 3 4))
```

- $\text{lon} \rightarrow \text{lon}$

```
; Purpose: Return a list of the squares of the given lon  
(define (square-lon a-lon)
```

- ;; Sample expressions for square-lon

```
(define LON1-VAL '())
```

```
(define LON2-VAL (cons (sqr (first '(1 2 3 4)))
                        (square-lon (rest '(1 2
```

Lists

Designing List-Processing Functions

- Write a function that squares a list of numbers

- ;; Sample instances of lon

```
(define LON1 '())
```

```
(define LON2 '(1 2 3 4))
```

- ;; lon → lon

- ;; Purpose: Return a list of the squares of the given lon

```
(define (square-lon a-lon)
```

- ;; Sample expressions for square-lon

```
(define LON1-VAL '())
```

```
(define LON2-VAL (cons (sqr (first '(1 2 3 4)))  
                        (square-lon (rest '(1 2 3 4)))))
```

- ;; Tests using sample computations for square-lon

```
(check-expect (square-lon LON1) LON1-VAL)
```

```
(check-expect (square-lon LON2) LON2-VAL)
```

```
;; Tests using sample values for square-lon
```

```
(check-expect (square-lon '(10 8 4)) '(100 64 16))
```

Lists

Designing List-Processing Functions

- Write a function that squares a list of numbers
 - ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```
  - ```
;; lon → lon
;; Purpose: Return a list of the squares of the given lon
(define (square-lon a-lon)
  (if (empty? a-lon)
      '()
```
 - ```
;; Sample expressions for square-lon
(define LON1-VAL '())
(define LON2-VAL (cons (sqr (first '(1 2 3 4)))
 (square-lon (rest '(1 2 3 4)))))
```
  - ```
;; Tests using sample computations for square-lon
(check-expect (square-lon LON1) LON1-VAL)
(check-expect (square-lon LON2) LON2-VAL)
```
- ```
;; Tests using sample values for square-lon
(check-expect (square-lon '(10 8 4)) '(100 64 16))
```

# Lists

## Designing List-Processing Functions

- Write a function that squares a list of numbers
  - ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```
 - ```
;; lon → lon
;; Purpose: Return a list of the squares of the given lon
(define (square-lon a-lon)
 (if (empty? a-lon)
 '()
 (cons (sqr (first a-lon)) (square-lon (rest a-lon))))))
```
  - ```
;; Sample expressions for square-lon
(define LON1-VAL '())
(define LON2-VAL (cons (sqr (first '(1 2 3 4)))
                        (square-lon (rest '(1 2 3 4)))))
```
 - ```
;; Tests using sample computations for square-lon
(check-expect (square-lon LON1) LON1-VAL)
(check-expect (square-lon LON2) LON2-VAL)
```
- ```
;; Tests using sample values for square-lon
(check-expect (square-lon '(10 8 4)) '(100 64 16))
```

Lists

Designing List-Processing Functions

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- There is a popular legend among students and professionals that recursion is hard

Lists

Designing List-Processing Functions

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- There is a popular legend among students and professionals that recursion is hard
- If you understand the design of `square-lon` then you know better

Lists

Designing List-Processing Functions

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- There is a popular legend among students and professionals that recursion is hard
- If you understand the design of `square-lon` then you know better
- Recursion is not hard nor should it be feared
- Understanding how to design solutions to problems based on data definitions makes recursion quite natural

Part III: Compound Data of Arbitrary Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists Homework

- Problems: 121–123

List Processing

- We explore common list operations:

`summarizing` : computes an aggregate value from the list

`searching` : list membership

`mapping` : apply a function to every list element and return a new list with the results

`filtering` : extract all the list elements that satisfy a property

`sorting` : list (ordinal or numerical) elements put in order

List Processing

List Summarizing

- How can you represent your quiz grades?
- Ask yourself if you know the number of quizzes in advance

List Processing

List Summarizing

- How can you represent your quiz grades?
- Ask yourself if you know the number of quizzes in advance
- There may be, for example, 0, 19, or 7 quizzes
- The point is that the number of quizzes is arbitrary

List Processing

List Summarizing

- How can you represent your quiz grades?
- Ask yourself if you know the number of quizzes in advance
- There may be, for example, 0, 19, or 7 quizzes
- The point is that the number of quizzes is arbitrary
- A first attempt to define a list of quiz grades:
A list of quiz grades (loq) is a (listof number).
- Is this a reasonable representation of quiz grades?

List Processing

List Summarizing

- How can you represent your quiz grades?
- Ask yourself if you know the number of quizzes in advance
- There may be, for example, 0, 19, or 7 quizzes
- The point is that the number of quizzes is arbitrary
- A first attempt to define a list of quiz grades:

A list of quiz grades (loq) is a (listof number).

- Is this a reasonable representation of quiz grades?
- Unfortunately, it is not:

```
'(87 65 92 -45 88 -7 98)
```

List Processing

List Summarizing

- How can you represent your quiz grades?
- Ask yourself if you know the number of quizzes in advance
- There may be, for example, 0, 19, or 7 quizzes
- The point is that the number of quizzes is arbitrary
- A first attempt to define a list of quiz grades:

```
A list of quiz grades (loq) is a (listof number).
```

- Is this a reasonable representation of quiz grades?
- Unfortunately, it is not:

```
'(87 65 92 -45 88 -7 98)
```

- Need to be specific about what a quiz grade is
- Assuming a quiz grade is based on a 100-point scale:

```
; ; A quiz grade (qg) is a number in [0..100]
```

```
; ; A list of quiz grades (loq) is a (listof qg)
```

```
; ; Sample values of loq
```

```
(define LOQ1 '())
```

```
(define LOQ2 '(87 65 92 88 98))
```

List Processing

List Summarizing

- **;; Sample instances of loq**

```
(define LOX1 ...)  
(define LOX2 ...)
```

```
;; loq ... → ...
```

```
;; Purpose: ...
```

```
(define (f-on-loq a-loq ...)  
  (if (empty? loq)  
      '()  
      ...  
      (f-on-qg (first a-loq))...  
      (f-on-loq (rest a-loq) ...)...)))
```

```
;; Sample expressions for f-on-loq
```

```
(define LOQ1-VAL ...)  
(define LOQ2-VAL ...) ...
```

```
;; Tests using sample computations for f-on-loq
```

```
(check-expect (f-on-loq LOQ1 ...) LOQ1-VAL)  
(check-expect (f-on-loq LOQ2 ...) LOQ2-VAL) ...
```

```
;; Tests using sample values for f-on-loq
```

```
(check-expect (f-on-loq ...) ...) ...
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Summarizing

- Write a function to compute the average of a list

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Summarizing

- Write a function to compute the average of a list
- How is a quiz average is computed?

List Processing

List Summarizing

- Write a function to compute the average of a list
- How is a quiz average is computed?
- - 1 Compute the length of the given list
 - 2 Compute the sum of the given list
 - 3 Divide the sum by the length
- Given that three different values are needed, how many different functions are needed?

List Processing

List Summarizing

- Write a function to compute the average of a list
- How is a quiz average is computed?
- - ① Compute the length of the given list
 - ② Compute the sum of the given list
 - ③ Divide the sum by the length
- Given that three different values are needed, how many different functions are needed?
- The most natural answer is three: one for each needed value
- This is underlined by the principle of separation of concerns

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Summarizing

- Start with the function that computes the average of a list
- Summarize a property of the given list

List Processing

List Summarizing

- Start with the function that computes the average of a `list`
- Summarize a property of the given `list`
- Divides its sum by its length
- This function does not process the given `list`
- It manipulates two list-summarizing values
- Using the template for a function on a `list` is incorrect

List Processing

List Summarizing

- Start with the function that computes the average of a list
- Summarize a property of the given list
- Divides its sum by its length
- This function does not process the given list
- It manipulates two list-summarizing values
- Using the template for a function on a list is incorrect
- The given list may not be empty
- Throw an error if the given list is empty

List Processing

List Summarizing

- Start with the function that computes the average of a `list`
- Summarize a property of the given `list`
- Divides its sum by its length
- This function does not process the given `list`
- It manipulates two list-summarizing values
- Using the template for a function on a `list` is incorrect
- The given `list` may not be empty
- Throw an error if the given `list` is empty
- The design assumes that functions to compute the sum and the length exist

List Processing

List Summarizing

- Start with the function that computes the average of a `loq`
- Summarize a property of the given `loq`
- Divides its sum by its length
- This function does not process the given `loq`
- It manipulates two list-summarizing values
- Using the template for a function on a `loq` is incorrect
- The given `loq` may not be empty
- Throw an error if the given `loq` is empty
- The design assumes that functions to compute the sum and the length exist

- `; ; Sample expressions for avg-loq`
`(define LOQ2-VAL (avg-loq LOQ2))`

List Processing

List Summarizing

- Start with the function that computes the average of a loq
- Summarize a property of the given loq
- Divides its sum by its length
- This function does not process the given loq
- It manipulates two list-summarizing values
- Using the template for a function on a loq is incorrect
- The given loq may not be empty
- Throw an error if the given loq is empty
- The design assumes that functions to compute the sum and the length exist
- `; ; loq → number throws error`
`; ; Purpose: To compute the given loq's average`
`(define (avg-loq a-loq)`

- `; ; Sample expressions for avg-loq`
`(define LOQ2-VAL (avg-loq LOQ2))`

List Processing

List Summarizing

- Start with the function that computes the average of a loq
- Summarize a property of the given loq
- Divides its sum by its length
- This function does not process the given loq
- It manipulates two list-summarizing values
- Using the template for a function on a loq is incorrect
- The given loq may not be empty
- Throw an error if the given loq is empty
- The design assumes that functions to compute the sum and the length exist
- `; ; loq → number throws error`
`; ; Purpose: To compute the given loq's average`
`(define (avg-loq a-loq)`

- `; ; Sample expressions for avg-loq`
`(define LOQ2-VAL (avg-loq LOQ2))`
- `; ; Tests using sample computations for avg-loq`
`(check-expect (avg-loq LOQ2) LOQ2-VAL)`

`; ; Tests using sample values for avg-loq`
`(check-error (avg-loq LOQ1) "avg-loq expects an non-empty.")`

List Processing

List Summarizing

- Start with the function that computes the average of a loq
- Summarize a property of the given loq
- Divides its sum by its length
- This function does not process the given loq
- It manipulates two list-summarizing values
- Using the template for a function on a loq is incorrect
- The given loq may not be empty
- Throw an error if the given loq is empty
- The design assumes that functions to compute the sum and the length exist
- ```
; ; loq → number throws error
; ; Purpose: To compute the given loq's average
(define (avg-loq a-loq)
 (if (empty? a-loq)
 (error (avg-loq expects an non-empty loq))
 (/ (sum-loq a-loq) (loq-len a-loq))))
```
- ```
; ; Sample expressions for avg-loq
(define LOQ2-VAL (avg-loq LOQ2))
```
- ```
; ; Tests using sample computations for avg-loq
(check-expect (avg-loq LOQ2) LOQ2-VAL)
```
- ```
; ; Tests using sample values for avg-loq
(check-error (avg-loq LOQ1) "avg-loq expects an non-empty.")
```

List Processing

List Summarizing

- Design the auxiliary function for the sum

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty list is 0
- Sum of an non-empty list: sum of the rest of list and first element

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty list is 0
- Sum of an non-empty list: sum of the rest of list and first element
- Observe that we statically reason about the list
- We reason about value obtained from first quiz grade and value obtained from the rest of the list

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty list is 0
- Sum of an non-empty list: sum of the rest of list and first element
- Observe that we statically reason about the list
- We reason about value obtained from first quiz grade and value obtained from the rest of the list
- ; Sample values of list
 - (define LOQ3 '(50 90 80 60 70))

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty loq is 0
- Sum of an non-empty loq: sum of the rest of loq and first element
- Observe that we statically reason about the loq
- We reason about value obtained from first quiz grade and value obtained from the rest of the list
- ;; Sample values of loq

```
(define LOQ3 '(50 90 80 60 70))
```

- ;; Sample expressions for sum-loq

```
(define SUM-LOQ1-VAL 0)
(define SUM-LOQ2-VAL (+ (first LOQ2) (sum-loq (rest LOQ2))))
(define SUM-LOQ3-VAL (+ (first LOQ3) (sum-loq (rest LOQ3))))
```

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty loq is 0
- Sum of an non-empty loq: sum of the rest of loq and first element
- Observe that we statically reason about the loq
- We reason about value obtained from first quiz grade and value obtained from the rest of the list
- ;; Sample values of loq

```
(define LOQ3 '(50 90 80 60 70))
```
- ;; loq → number Purpose: To compute the sum of the given loq

```
(define (sum-loq a-loq)
```
- ;; Sample expressions for sum-loq

```
(define SUM-LOQ1-VAL 0)
(define SUM-LOQ2-VAL (+ (first LOQ2) (sum-loq (rest LOQ2))))
(define SUM-LOQ3-VAL (+ (first LOQ3) (sum-loq (rest LOQ3))))
```

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty loq is 0
- Sum of an non-empty loq: sum of the rest of loq and first element
- Observe that we statically reason about the loq
- We reason about value obtained from first quiz grade and value obtained from the rest of the list
- ```
;; Sample values of loq
(define LOQ3 '(50 90 80 60 70))
```
- ```
;; loq → number Purpose: To compute the sum of the given loq
(define (sum-loq a-loq)
```

- ```
;; Sample expressions for sum-loq
(define SUM-LOQ1-VAL 0)
(define SUM-LOQ2-VAL (+ (first LOQ2) (sum-loq (rest LOQ2))))
(define SUM-LOQ3-VAL (+ (first LOQ3) (sum-loq (rest LOQ3))))
```
- ```
;; Tests using sample computations for sum-loq
(check-expect (sum-loq LOQ1) SUM-LOQ1-VAL)
(check-expect (sum-loq LOQ2) SUM-LOQ2-VAL)
;; Tests using sample values for sum-loq
(check-expect (sum-loq '(96 97 99 100)) 392)
```

List Processing

List Summarizing

- Design the auxiliary function for the sum
- The sum of an empty loq is 0
- Sum of an non-empty loq: sum of the rest of loq and first element
- Observe that we statically reason about the loq
- We reason about value obtained from first quiz grade and value obtained from the rest of the list
- ```
;; Sample values of loq
(define LOQ3 '(50 90 80 60 70))
```
- ```
;; loq → number Purpose: To compute the sum of the given loq
(define (sum-loq a-loq)
  (if (empty? a-loq)
      0
      (+ (first a-loq) (sum-loq (rest a-loq)))))
```
- ```
;; Sample expressions for sum-loq
(define SUM-LOQ1-VAL 0)
(define SUM-LOQ2-VAL (+ (first LOQ2) (sum-loq (rest LOQ2))))
(define SUM-LOQ3-VAL (+ (first LOQ3) (sum-loq (rest LOQ3))))
```
- ```
;; Tests using sample computations for sum-loq
(check-expect (sum-loq LOQ1) SUM-LOQ1-VAL)
(check-expect (sum-loq LOQ2) SUM-LOQ2-VAL)
;; Tests using sample values for sum-loq
(check-expect (sum-loq '(96 97 99 100)) 392)
```

List Processing

List Summarizing

- Design a function to compute the length of a list

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Summarizing

- Design a function to compute the length of a list
- The length is 0 if the given list is empty
- The length is 1 + the length of the rest of list if the given list is not empty

List Processing

List Summarizing

- Design a function to compute the length of a list
- The length is 0 if the given list is empty
- The length is 1 + the length of the rest of list if the given list is not empty

List Processing

List Summarizing

- Design a function to compute the length of a loq
- The length is 0 if the given loq is empty
- The length is 1 + the length of the rest of loq if the given loq is not empty

- ```
;; Sample expressions for length-loq
(define LEN-LOQ1-VAL 0)
(define LEN-LOQ2-VAL (+ 1 (length-loq (rest LOQ2))))
(define LEN-LOQ3-VAL (+ 1 (length-loq (rest LOQ3))))
```

# List Processing

## List Summarizing

- Design a function to compute the length of a loq
- The length is 0 if the given loq is empty
- The length is 1 + the length of the rest of loq if the given loq is not empty
- ```
;; loq → number
;; Purpose: To compute the length of the given loq
(define (length-loq a-loq))
```
- ```
;; Sample expressions for length-loq
(define LEN-LOQ1-VAL 0)
(define LEN-LOQ2-VAL (+ 1 (length-loq (rest LOQ2))))
(define LEN-LOQ3-VAL (+ 1 (length-loq (rest LOQ3))))
```

# List Processing

## List Summarizing

- Design a function to compute the length of a loq
- The length is 0 if the given loq is empty
- The length is 1 + the length of the rest of loq if the given loq is not empty
- ```
;; loq → number
;; Purpose: To compute the length of the given loq
(define (length-loq a-loq))
```

- ```
;; Sample expressions for length-loq
(define LEN-LOQ1-VAL 0)
(define LEN-LOQ2-VAL (+ 1 (length-loq (rest LOQ2))))
(define LEN-LOQ3-VAL (+ 1 (length-loq (rest LOQ3))))
```
  - ```
;; Tests using sample computations for length-loq
(check-expect (length-loq LOQ1) LEN-LOQ1-VAL)
(check-expect (length-loq LOQ2) LEN-LOQ2-VAL)
(check-expect (length-loq LOQ3) LEN-LOQ3-VAL)
```
- ```
;; Tests using sample values for length-loq
(check-expect (length-loq '(96 97 99 100 89 94)) 6)
```

# List Processing

## List Summarizing

- Design a function to compute the length of a loq
- The length is 0 if the given loq is empty
- The length is 1 + the length of the rest of loq if the given loq is not empty
- ```
;; loq → number
;; Purpose: To compute the length of the given loq
(define (length-loq a-loq))
```
- ```
(if (empty? a-loq)
 0
 (+ 1 (length-loq (rest a-loq)))))
```
- ```
;; Sample expressions for length-loq
(define LEN-LOQ1-VAL 0)
(define LEN-LOQ2-VAL (+ 1 (length-loq (rest LOQ2))))
(define LEN-LOQ3-VAL (+ 1 (length-loq (rest LOQ3))))
```
- ```
;; Tests using sample computations for length-loq
(check-expect (length-loq LOQ1) LEN-LOQ1-VAL)
(check-expect (length-loq LOQ2) LEN-LOQ2-VAL)
(check-expect (length-loq LOQ3) LEN-LOQ3-VAL)
```
- ```
;; Tests using sample values for length-loq
(check-expect (length-loq '(96 97 99 100 89 94)) 6)
```

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

Homework

- Problems: 125–130

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Searching

- Another common operation is searching a given list for a value

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Searching

- Another common operation is searching a given list for a value
- For example, given a list of numbers, $a\text{-lon}$, and a number, x , you need the sublist of $a\text{-lon}$ that starts with the first occurrence of x

List Processing

List Searching

- Another common operation is searching a given list for a value
- For example, given a list of numbers, `a-lon`, and a number, `x`, you need the sublist of `a-lon` that starts with the first occurrence of `x`
- If `a-lon` is empty the answer is `'()'` because `a-lon` does not contain `x`
- What if the list is not empty?

List Processing

List Searching

- Another common operation is searching a given list for a value
- For example, given a list of numbers, `a-lon`, and a number, `x`, you need the sublist of `a-lon` that starts with the first occurrence of `x`
- If `a-lon` is empty the answer is `'()` because `a-lon` does not contain `x`
- What if the list is not empty?
- Observe that `x` may be the first element of the list: the answer is `a-lon`

List Processing

List Searching

- Another common operation is searching a given list for a value
- For example, given a list of numbers, `a-lon`, and a number, `x`, you need the sublist of `a-lon` that starts with the first occurrence of `x`
- If `a-lon` is empty the answer is '`()`' because `a-lon` does not contain `x`
- What if the list is not empty?
- Observe that `x` may be the first element of the list: the answer is `a-lon`
- If `(first a-lon)` is not equal to `x`: `(rest a-lon)` must be recursively searched

List Processing

List Searching

- Another common operation is searching a given list for a value
- For example, given a list of numbers, `a-lon`, and a number, `x`, you need the sublist of `a-lon` that starts with the first occurrence of `x`
- If `a-lon` is empty the answer is `'()` because `a-lon` does not contain `x`
- What if the list is not empty?
- Observe that `x` may be the first element of the list: the answer is `a-lon`
- If `(first a-lon)` is not equal to `x`: `(rest a-lon)` must be recursively searched
- There are three conditions that need to be detected (not two as suggested by the template for a function on a `lon`)

List Processing

List Searching

- ; ; Sample instances of lon
`(define LON1 '())
(define LON2 '(1 2 3 4))`

List Processing

List Searching

- ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```

- ```
;; Sample expressions for xs sublist-lon
(define XSUBLIST-LON1-VAL '())
(define XSUBLIST-LON2-VAL1 LON2)
(define XSUBLIST-LON2-VAL2 (xsublist-lon 3 (rest LON2)))
(define XSUBLIST-LON2-VAL3 (xsublist-lon 0 (rest LON2)))
```

List Processing

List Searching

- ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```
- ```
;; number lon → lon
;; Purpose: Return the sublist that starts with the
;;           first instance of the given number.
(define (xsublist-lon x a-lon)
```
- ```
;; Sample expressions for xsublist-lon
(define XSUBLIST-LON1-VAL '())
(define XSUBLIST-LON2-VAL1 LON2)
(define XSUBLIST-LON2-VAL2 (xsublist-lon 3 (rest LON2)))
(define XSUBLIST-LON2-VAL3 (xsublist-lon 0 (rest LON2)))
```

# List Processing

## List Searching

- ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```
- ```
;; number lon → lon
;; Purpose: Return the sublist that starts with the
;; first instance of the given number.
(define (xsublist-lon x a-lon)
```
- ```
;; Sample expressions for xsublist-lon
(define XSUBLIST-LON1-VAL '())
(define XSUBLIST-LON2-VAL1 LON2)
(define XSUBLIST-LON2-VAL2 (xsublist-lon 3 (rest LON2)))
(define XSUBLIST-LON2-VAL3 (xsublist-lon 0 (rest LON2)))
```
- ```
;; Tests using sample computations for xsublist-lon
(check-expect (xsublist-lon 7 LON1) XSUBLIST-LON1-VAL)
(check-expect (xsublist-lon 1 LON2) XSUBLIST-LON2-VAL1)
(check-expect (xsublist-lon 3 LON2) XSUBLIST-LON2-VAL2)
(check-expect (xsublist-lon 0 LON2) XSUBLIST-LON2-VAL3)
;; Tests using sample values for xsublist-lon
(check-expect (xsublist-lon 99 '(96 97 99 100)) '(99 100))
(check-expect (xsublist-lon 87 '(86 47 10)) '())
```

# List Processing

## List Searching

- ```
;; Sample instances of lon
(define LON1 '())
(define LON2 '(1 2 3 4))
```
- ```
;; number lon → lon
;; Purpose: Return the sublist that starts with the
;; first instance of the given number.
(define (xsublist-lon x a-lon)
 (cond [(empty? a-lon) '()]
 [(equal? x (first a-lon)) a-lon]
 [else (xsublist-lon x (rest a-lon))]))
```
- ```
;; Sample expressions for xsublist-lon
(define XSUBLIST-LON1-VAL '())
(define XSUBLIST-LON2-VAL1 LON2)
(define XSUBLIST-LON2-VAL2 (xsublist-lon 3 (rest LON2)))
(define XSUBLIST-LON2-VAL3 (xsublist-lon 0 (rest LON2)))
```
- ```
;; Tests using sample computations for xsublist-lon
(check-expect (xsublist-lon 7 LON1) XSUBLIST-LON1-VAL)
(check-expect (xsublist-lon 1 LON2) XSUBLIST-LON2-VAL1)
(check-expect (xsublist-lon 3 LON2) XSUBLIST-LON2-VAL2)
(check-expect (xsublist-lon 0 LON2) XSUBLIST-LON2-VAL3)
;; Tests using sample values for xsublist-lon
(check-expect (xsublist-lon 99 '(96 97 99 100)) '(99 100))
(check-expect (xsublist-lon 87 '(86 47 10)) '())
```

**Part III:**  
**Compound**  
**Data of**  
**Arbitrary**  
**Size**

**Marco T.**  
**Morazán**

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## Homework

- Problems: 131–133

# List Processing

## List ORing

- A variant of list searching is list ORing
- ORing determines if there exists a list value in a given list, L, that satisfies some property P
- What is the answer if L is empty?

# List Processing

## List ORing

- A variant of list searching is list ORing
- ORing determines if there exists a list value in a given list, L, that satisfies some property P
- What is the answer if L is empty?
- The answer may not be clear to you: let us delay the answer for now

# List Processing

## List ORing

- A variant of list searching is list ORing
- ORing determines if there exists a list value in a given list, L, that satisfies some property P
- What is the answer if L is empty?
- The answer may not be clear to you: let us delay the answer for now
- Think about where p does not satisfy P:

$L = (\text{cons } p \text{ '()})$

- Is there a an element in L that satisfies P?
- Clearly the answer is no
- Think about how L is processed

# List Processing

## List ORing

- A variant of list searching is list ORing
- ORing determines if there exists a list value in a given list, L, that satisfies some property P
- What is the answer if L is empty?
- The answer may not be clear to you: let us delay the answer for now
- Think about where p does not satisfy P:

`L = (cons p '())`

- Is there a an element in L that satisfies P?
- Clearly the answer is no
- Think about how L is processed
- You need to apply P to p to obtain #false
- This result must be ored with the result obtained from recursively processing the rest of L

# List Processing

## List ORing

- A variant of list searching is list ORing
- ORing determines if there exists a list value in a given list, L, that satisfies some property P
- What is the answer if L is empty?
- The answer may not be clear to you: let us delay the answer for now
- Think about where p does not satisfy P:

$L = (\text{cons } p \text{ '()})$

- Is there a an element in L that satisfies P?
- Clearly the answer is no
- Think about how L is processed
- You need to apply P to p to obtain #false
- This result must be ored with the result obtained from recursively processing the rest of L
- This means that '() must be recursively processed
- The only way processing  $(\text{cons } p \text{ '()})$  returns #false is if processing '() returns #false

# List Processing

## List ORing

- A variant of list searching is list ORing
- ORing determines if there exists a list value in a given list, L, that satisfies some property P
- What is the answer if L is empty?
- The answer may not be clear to you: let us delay the answer for now
- Think about where p does not satisfy P:

$L = (\text{cons } p \text{ '()})$

- Is there a an element in L that satisfies P?
- Clearly the answer is no
- Think about how L is processed
- You need to apply P to p to obtain #false
- This result must be ored with the result obtained from recursively processing the rest of L
- This means that '() must be recursively processed
- The only way processing  $(\text{cons } p \text{ '()})$  returns #false is if processing '() returns #false
- This means that the answer is #false when L is empty
- If L is not empty then you must apply or to the result of applying P to p and of recursively processing the rest of L

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ORing

- In Aliens Attack all the aliens move in the same direction
- The direction only changes when there exists an alien that is either at the left or the right edge of the scene
- We need predicates to determine if there exists an alien located at either the right or left edge in a list of aliens

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ORing

- Think about what it means for their to exist an alien that is at the left edge of the scene

# List Processing

## List ORing

- Think about what it means for there to exist an alien that is at the left edge of the scene
- One condition is that the given list of aliens cannot be empty
- This is a *necessary* condition for such an alien to exist, but it is not *sufficient*
- A necessary condition is one that must be true for a predicate to be true, but is not enough to conclude that the predicate is true
- A sufficient condition is one that allows us to conclude that the predicate is true
- What other condition(s) must hold?
- Think in terms of the structure of the list you need to process

# List Processing

## List ORing

- Think about what it means for there to exist an alien that is at the left edge of the scene
- One condition is that the given list of aliens cannot be empty
- This is a *necessary* condition for such an alien to exist, but it is not *sufficient*
- A necessary condition is one that must be true for a predicate to be true, but is not enough to conclude that the predicate is true
- A sufficient condition is one that allows us to conclude that the predicate is true
- What other condition(s) must hold?
- Think in terms of the structure of the list you need to process
- Either the first alien is at the left edge or an alien in the rest of the list of aliens is at the left edge

# List Processing

## List ORing

- Think about what it means for there to exist an alien that is at the left edge of the scene
- One condition is that the given list of aliens cannot be empty
- This is a *necessary* condition for such an alien to exist, but it is not *sufficient*
- A necessary condition is one that must be true for a predicate to be true, but is not enough to conclude that the predicate is true
- A sufficient condition is one that allows us to conclude that the predicate is true
- What other condition(s) must hold?
- Think in terms of the structure of the list you need to process
- Either the first alien is at the left edge or an alien in the rest of the list of aliens is at the left edge
- Observe that no decision must be made to compute this value
- That is, a conditional as suggested by the template for a function on a list of aliens is not needed
- Use `and` to detect if both conditions hold
- Use `or` to detect if the first alien is at the left edge or if any alien in the rest of the list is at the left edge

# List Processing

## List ORing

- Samples for testing:

```
(define ALIEN-8-0 (make-posn 8 0))

;; Sample instances of loa
(define EDGE-LOA (list ALIEN-8-0
 (make-posn MIN-IMG-X 11)
 (make-posn 5 5)))
(define EDGE-LOA2 (list (make-posn 1 11)
 (make-posn MAX-IMG-X 5)))
```

# List Processing

## List ORing

- ;; Sample expressions for any-alien-at-left-edge?  

```
(define LEDGE-E-LOA-VAL alien-at-left-edge? in Aliens Attack 2
 (and (not (empty? E-LOA))
 (or (alien-at-left-edge? (first E-LOA))
 (any-alien-at-left-edge? (rest E-LOA)))))

(define LEDGE-INIT-LOA-VAL
 (and (not (empty? INIT-LOA))
 (or (alien-at-left-edge? (first INIT-LOA))
 (any-alien-at-left-edge? (rest INIT-LOA)))))

(define LEDGE-LOA-VAL
 (and (not (empty? EDGE-LOA))
 (or (alien-at-left-edge? (first EDGE-LOA))
 (any-alien-at-left-edge? (rest EDGE-LOA)))))

(define LEDGE-LOA2-VAL
 (and (not (empty? EDGE-LOA2))
 (or (alien-at-left-edge? (first EDGE-LOA2))
 (any-alien-at-left-edge? (rest EDGE-LOA2)))))
```

# List Processing

## List ORing

- `;; loa → Boolean Purpose: Determine if any alien at left edge`  
`(define (any-alien-at-left-edge? a-loa)`

- `;; Sample expressions for any-alien-at-left-edge?`  
`(define LEDGE-E-LOA-VAL alien-at-left-edge? in Aliens Attack 2`  
 `(and (not (empty? E-LOA))`  
 `(or (alien-at-left-edge? (first E-LOA))`  
 `(any-alien-at-left-edge? (rest E-LOA))))))`
- `(define LEDGE-INIT-LOA-VAL`  
 `(and (not (empty? INIT-LOA))`  
 `(or (alien-at-left-edge? (first INIT-LOA))`  
 `(any-alien-at-left-edge? (rest INIT-LOA))))))`
- `(define LEDGE-LOA-VAL`  
 `(and (not (empty? EDGE-LOA))`  
 `(or (alien-at-left-edge? (first EDGE-LOA))`  
 `(any-alien-at-left-edge? (rest EDGE-LOA))))))`
- `(define LEDGE-LOA2-VAL`  
 `(and (not (empty? EDGE-LOA2))`  
 `(or (alien-at-left-edge? (first EDGE-LOA2))`  
 `(any-alien-at-left-edge? (rest EDGE-LOA2))))))`

# List Processing

## List ORing

- `;; loa → Boolean Purpose: Determine if any alien at left edge`  
`(define (any-alien-at-left-edge? a-loa)`  
 `(and (not (empty? a-loa))`  
 `(or (alien-at-left-edge? (first a-loa))`  
 `(any-alien-at-left-edge? (rest a-loa))))))`
- `;; Sample expressions for any-alien-at-left-edge?`  
`(define LEDGE-E-LOA-VAL alien-at-left-edge? in Aliens Attack 2`  
 `(and (not (empty? E-LOA))`  
 `(or (alien-at-left-edge? (first E-LOA))`  
 `(any-alien-at-left-edge? (rest E-LOA))))))`
- `(define LEDGE-INIT-LOA-VAL`  
 `(and (not (empty? INIT-LOA))`  
 `(or (alien-at-left-edge? (first INIT-LOA))`  
 `(any-alien-at-left-edge? (rest INIT-LOA))))))`
- `(define LEDGE-LOA-VAL`  
 `(and (not (empty? EDGE-LOA))`  
 `(or (alien-at-left-edge? (first EDGE-LOA))`  
 `(any-alien-at-left-edge? (rest EDGE-LOA))))))`
- `(define LEDGE-LOA2-VAL`  
 `(and (not (empty? EDGE-LOA2))`  
 `(or (alien-at-left-edge? (first EDGE-LOA2))`  
 `(any-alien-at-left-edge? (rest EDGE-LOA2))))))`

# List Processing

## List ORing

- ;; Tests using sample computations any-alien-at-left-edge?  
(check-expect (any-alien-at-left-edge? E-LOA) LEDGE-E-LOA-VAL)  
(check-expect (any-alien-at-left-edge? INIT-LOA)  
                 LEDGE-INIT-LOA-VAL)  
(check-expect (any-alien-at-left-edge? EDGE-LOA) LEDGE-LOA-VAL)  
(check-expect (any-alien-at-left-edge? EDGE-LOA2) LEDGE-LOA2-VAL)

# List Processing

## List ORing

- ;; Tests using sample computations any-alien-at-left-edge?  
(check-expect (any-alien-at-left-edge? E-LOA) LEDGE-E-LOA-VAL)  
(check-expect (any-alien-at-left-edge? INIT-LOA)  
                 LEDGE-INIT-LOA-VAL)  
(check-expect (any-alien-at-left-edge? EDGE-LOA) LEDGE-LOA-VAL)  
(check-expect (any-alien-at-left-edge? EDGE-LOA2) LEDGE-LOA2-VAL)
- ;; Tests using sample values for any-alien-at-left-edge?  
(check-expect  
  (any-alien-at-left-edge? (list (make-posn MIN-IMG-X 8)  
                              (make-posn 6 3)  
                              (make-posn MAX-IMG-X 10)))  
    #true)  
(check-expect  
  (any-alien-at-left-edge? (list (make-posn 3 8)  
                              (make-posn MIN-IMG-X 3)  
                              (make-posn 5 2)))  
    #true)  
(check-expect  
  (any-alien-at-left-edge? (list (make-posn MAX-IMG-Y 8))) #false)  
(check-expect  
  (any-alien-at-left-edge? (list (make-posn 3 8) (make-posn 5 2)))  
    #false)

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ORing

- Similar for `any-alien-at-right-edge?` and `any-alien-reached-earth?`
- Make sure to read the textbook

**Part III:**  
**Compound**  
**Data of**  
**Arbitrary**  
**Size**

**Marco T.**  
**Morazán**

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## Homework

- Problems: 134–138

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Another variant of list searching determines if all of L's elements satisfy a predicate P

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Another variant of list searching determines if all of L's elements satisfy a predicate P
- Consider a list of length 1 whose only element, p, satisfies P
- Do all the list elements satisfy P?

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Another variant of list searching determines if all of L's elements satisfy a predicate P
- Consider a list of length 1 whose only element, p, satisfies P
- Do all the list elements satisfy P?
- Clearly, the answer is yes
- $L = (\text{cons } p \text{ '()})$
- To process p you apply P to obtain #true
- The rest of the list is processed recursively
- Given that the rest of L is '(), the recursive call must evaluate to #true

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Another variant of list searching determines if all of L's elements satisfy a predicate P
- Consider a list of length 1 whose only element, p, satisfies P
- Do all the list elements satisfy *P*?
- Clearly, the answer is yes
- $L = (\text{cons } p \text{ '()})$
- To process p you apply P to obtain #true
- The rest of the list is processed recursively
- Given that the rest of L is '()', the recursive call must evaluate to #true
- The answer must be #true when L is empty
- When the list is not empty and the result of applying P to the first element and the result obtained from recursively processing the rest of the list

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Consider determining if all the numbers in a list of numbers are even

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Consider determining if all the numbers in a list of numbers are even
- If the given `lon` is empty then the answer is true

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Consider determining if all the numbers in a list of numbers are even
- If the given `lon` is empty then the answer is true
- If the given `lon` is not empty then the first number must be even and the rest of the numbers must be even

# List Processing

## List ANDing

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Consider determining if all the numbers in a list of numbers are even
- If the given `lon` is empty then the answer is true
- If the given `lon` is not empty then the first number must be even and the rest of the numbers must be even
- Observe that no decision must be made to solve this problem and a condition expression as suggested by the template for a function on a `lon` is not needed

# List Processing

## List ANDing

- ;; Sample instances of lon  
`(define E-LON '()) (define AE-LON '(88 98 22 78 506))  
(define NAE-LON '(8 561 683 788))`

# List Processing

## List ANDing

- ;; Sample instances of lon  
(define E-LON '()) (define AE-LON '(88 98 22 78 506))  
(define NAE-LON '(8 561 683 788))

- ;; Sample expressions for all-even-lon?  
(define E-LON-VAL  
 (or (empty? E-LON)  
 (and (even? (first E-LON)) (all-even-lon? (rest E-LON)))))  
(define AE-LON-VAL  
 (or (empty? AE-LON)  
 (and (even? (first AE-LON)) (all-even-lon? (rest AE-LON)))))  
(define NAE-LON-VAL  
 (or (empty? NAE-LON)  
 (and (even? (first NAE-LON)) (all-even-lon? (rest NAE-LON)))))

# List Processing

## List ANDing

- ;; Sample instances of lon  
`(define E-LON '()) (define AE-LON '(88 98 22 78 506))  
(define NAE-LON '(8 561 683 788))`
- ;; lon → Boolean Purpose: Determine if lon only has even numbers  
`(define (all-even-lon? a-lon)`
- ;; Sample expressions for all-even-lon?  
`(define E-LON-VAL  
 (or (empty? E-LON)  
 (and (even? (first E-LON)) (all-even-lon? (rest E-LON)))))  
(define AE-LON-VAL  
 (or (empty? AE-LON)  
 (and (even? (first AE-LON)) (all-even-lon? (rest AE-LON)))))  
(define NAE-LON-VAL  
 (or (empty? NAE-LON)  
 (and (even? (first NAE-LON)) (all-even-lon? (rest NAE-LON)))))`

# List Processing

## List ANDing

- ;; Sample instances of lon  
`(define E-LON '()) (define AE-LON '(88 98 22 78 506))  
(define NAE-LON '(8 561 683 788))`
- ;; lon → Boolean Purpose: Determine if lon only has even numbers  
`(define (all-even-lon? a-lon)`
- ;; Sample expressions for all-even-lon?  
`(define E-LON-VAL  
(or (empty? E-LON)  
(and (even? (first E-LON)) (all-even-lon? (rest E-LON))))))  
(define AE-LON-VAL  
(or (empty? AE-LON)  
(and (even? (first AE-LON)) (all-even-lon? (rest AE-LON))))))  
(define NAE-LON-VAL  
(or (empty? NAE-LON)  
(and (even? (first NAE-LON)) (all-even-lon? (rest NAE-LON))))))`
- ;; Tests using sample computations for all-even-lon?  
`(check-expect (all-even-lon? E-LON) E-LON-VAL)  
(check-expect (all-even-lon? AE-LON) AE-LON-VAL)  
(check-expect (all-even-lon? NAE-LON) NAE-LON-VAL)  
;; Tests using sample values for all-even-lon?  
(check-expect (all-even-lon? '(9 1 7)) #false)  
(check-expect (all-even-lon? '((4 6 8))) #true)`

# List Processing

## List ANDing

- ;; Sample instances of lon  
`(define E-LON '()) (define AE-LON '(88 98 22 78 506))  
(define NAE-LON '(8 561 683 788))`
- ;; lon → Boolean Purpose: Determine if lon only has even numbers  
`(define (all-even-lon? a-lon)  
 (or (empty? a-lon)  
 (and (even? (first a-lon)) (all-even-lon? (rest a-lon))))))`
- ;; Sample expressions for all-even-lon?  
`(define E-LON-VAL  
 (or (empty? E-LON)  
 (and (even? (first E-LON)) (all-even-lon? (rest E-LON))))))  
(define AE-LON-VAL  
 (or (empty? AE-LON)  
 (and (even? (first AE-LON)) (all-even-lon? (rest AE-LON))))))  
(define NAE-LON-VAL  
 (or (empty? NAE-LON)  
 (and (even? (first NAE-LON)) (all-even-lon? (rest NAE-LON))))))`
- ;; Tests using sample computations for all-even-lon?  
`(check-expect (all-even-lon? E-LON) E-LON-VAL)  
(check-expect (all-even-lon? AE-LON) AE-LON-VAL)  
(check-expect (all-even-lon? NAE-LON) NAE-LON-VAL)  
;; Tests using sample values for all-even-lon?  
(check-expect (all-even-lon? '(9 1 7)) #false)  
(check-expect (all-even-lon? '((4 6 8))) #true)`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Consider the problem of determining if a list of numbers is sorted in non-decreasing order
- How do you know if a list is or is not in non-decreasing order?

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List ANDing

- Consider the problem of determining if a list of numbers is sorted in non-decreasing order
- How do you know if a list is or is not in non-decreasing order?
- As you traverse the list from left to right the first number must be less than or equal to the second number
- What if the list does not have two numbers?

# List Processing

## List ANDing

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Consider the problem of determining if a list of numbers is sorted in non-decreasing order
- How do you know if a list is or is not in non-decreasing order?
- As you traverse the list from left to right the first number must be less than or equal to the second number
- What if the list does not have two numbers?
- The list is sorted in non-decreasing order.

# List Processing

## List ANDing

- Consider the problem of determining if a list of numbers is sorted in non-decreasing order
- How do you know if a lon is or is not in non-decreasing order?
- As you traverse the list from left to right the first number must be less than or equal to the second number
- What if the lon does not have two numbers?
- The lon is sorted in non-decreasing order.
- Data analysis suggests we need a different data definition for a list of numbers:

```
;; A lon2 is either:
;; 1. ()
;; 2. (list number)
;; 3. (list number number lon2)
```

# List Processing

## List ANDing

- ```
;; Sample instances of lon2
;; (define E-LON2 ...) (define A-LON2 ...) (define B-LON2 ...)

;; lon2 ... --> ... Purpose: ...
;; (define (f-on-lon2 a-lon2 ...)
;;   (cond [(empty? a-lon2) ...]
;;         [=(length a-lon2) 1) ...]
;;         [else ...(f-on-number (first a-lon2))
;;           ... (f-on-number (second a-lon2))
;;           ... (f-on-lon2 (rest a-lon2))
;;           ... (f-on-lon2 (rest (rest a-lon2))))])

;; Sample expressions for sorted-lon2
;; (define E-LON2-VAL ...)
;; (define A-LON2-VAL ...)
;; (define B-LON2-VAL ...) ...

;; Tests using sample computations for sorted-LON2
;; (check-expect (sorted-lon2? E-LON2 ...) E-LON2-VAL)
;; (check-expect (sorted-lon2? A-LON2 ...) A-LON2-VAL)
;; (check-expect (sorted-lon2? B-LON2 ...) B-LON2-VAL) ...

;; Tests using sample values for sorted-lon2
;; (check-expect (sorted-lon2? ...) ...) ...
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List ANDing

- If the given `lon2` is empty then the numbers are sorted in non-decreasing order

List Processing

List ANDing

- If the given `lon2` is empty then the numbers are sorted in non-decreasing order
- The same is true if the given `lon2` only has one number
- This suggests that the first two varieties may be combined in our function

List Processing

List ANDing

- If the given `lon2` is empty then the numbers are sorted in non-decreasing order
- The same is true if the given `lon2` only has one number
- This suggests that the first two varieties may be combined in our function
- If the given `lon2` has two or more elements then the first two elements must be in non-decreasing order
- The recursive call is made with the rest of the given `lon2`.
- Observe that a decision must not be made to compute the needed answer
- We can or the result of determining if the list is of at most length 1 and the result of anding the result of comparing the first two numbers and the result of recursively processing the rest of the given `lon2`

List Processing

List ANDing

- ; Sample instances of lon2
(define E-LON2 '()) (define A-LON2 '(88))
(define SORTED-LON2 '(8 56 68 788)) (define UNSORTED-LON2 '(5 7 8 4 9))

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List ANDing

- ; Sample instances of lon2
`(define E-LON2 '()) (define A-LON2 '(88))
(define SORTED-LON2 '(8 56 68 788)) (define UNSORTED-LON2 '(5 7 8 4 9))`

- ; Sample expressions for sorted-lon2
`(define E-LON2-VAL
 (or (< (length E-LON2) 2)
 (and (≤ (first E-LON2) (second E-LON2)) (sorted-lon2? (rest E-LON2)))))
(define A-LON2-VAL
 (or (< (length A-LON2) 2)
 (and (≤ (first A-LON2) (second A-LON2)) (sorted-lon2? (rest A-LON2)))))
(define SORTED-LON2-VAL
 (or (< (length SORTED-LON2) 2)
 (and (≤ (first SORTED-LON2) (second SORTED-LON2))
 (sorted-lon2? (rest SORTED-LON2)))))
(define UNSORTED-LON2-VAL
 (or (< (length UNSORTED-LON2) 2)
 (and (≤ (first UNSORTED-LON2)
 (second UNSORTED-LON2))
 (sorted-lon2? (rest UNSORTED-LON2))))))`

List Processing

List ANDing

- ; Sample instances of lon2
`(define E-LON2 '()) (define A-LON2 '(88))
(define SORTED-LON2 '(8 56 68 788)) (define UNSORTED-LON2 '(5 7 8 4 9))`
- ; lon2 → Boolean Purpose: Determine if given lon2 is in nondecreasing order
`(define (sorted-lon2? a-lon2)`
- ; Sample expressions for sorted-lon2
`(define E-LON2-VAL
 (or (< (length E-LON2) 2)
 (and (≤ (first E-LON2) (second E-LON2)) (sorted-lon2? (rest E-LON2))))))
(define A-LON2-VAL
 (or (< (length A-LON2) 2)
 (and (≤ (first A-LON2) (second A-LON2)) (sorted-lon2? (rest A-LON2))))))
(define SORTED-LON2-VAL
 (or (< (length SORTED-LON2) 2)
 (and (≤ (first SORTED-LON2) (second SORTED-LON2))
 (sorted-lon2? (rest SORTED-LON2))))))
(define UNSORTED-LON2-VAL
 (or (< (length UNSORTED-LON2) 2)
 (and (≤ (first UNSORTED-LON2)
 (second UNSORTED-LON2))
 (sorted-lon2? (rest UNSORTED-LON2))))))`

List Processing

List ANDing

- ; Sample instances of lon2
(define E-LON2 '()) (define A-LON2 '(88))
(define SORTED-LON2 '(8 56 68 788)) (define UNSORTED-LON2 '(5 7 8 4 9))
- ; lon2 → Boolean Purpose: Determine if given lon2 is in nondecreasing order
(define (sorted-lon2? a-lon2)
- ; Sample expressions for sorted-lon2
(define E-LON2-VAL
 (or (< (length E-LON2) 2)
 (and (≤ (first E-LON2) (second E-LON2)) (sorted-lon2? (rest E-LON2))))))
(define A-LON2-VAL
 (or (< (length A-LON2) 2)
 (and (≤ (first A-LON2) (second A-LON2)) (sorted-lon2? (rest A-LON2))))))
(define SORTED-LON2-VAL
 (or (< (length SORTED-LON2) 2)
 (and (≤ (first SORTED-LON2) (second SORTED-LON2))
 (sorted-lon2? (rest SORTED-LON2))))))
(define UNSORTED-LON2-VAL
 (or (< (length UNSORTED-LON2) 2)
 (and (≤ (first UNSORTED-LON2)
 (second UNSORTED-LON2))
 (sorted-lon2? (rest UNSORTED-LON2))))))
- ; Tests using sample computations for sorted-LON2
(check-expect (sorted-lon2? E-LON2) E-LON2-VAL)
(check-expect (sorted-lon2? A-LON2) A-LON2-VAL)
(check-expect (sorted-lon2? SORTED-LON2) SORTED-LON2-VAL)
(check-expect (sorted-lon2? UNSORTED-LON2) UNSORTED-LON2-VAL)
;; Tests using sample values for sorted-lon2
(check-expect (sorted-lon2? '(8 7 6 5)) #false)
(check-expect (sorted-lon2? '(5 6 7 8)) #true)

List Processing

List ANDing

- ; Sample instances of lon2
(define E-LON2 '()) (define A-LON2 '(88))
(define SORTED-LON2 '(8 56 68 788)) (define UNSORTED-LON2 '(5 7 8 4 9))
- ; lon2 → Boolean Purpose: Determine if given lon2 is in nondecreasing order
(define (sorted-lon2? a-lon2)
 (or (<= 0 (length a-lon2) 1)
 (and (<= (first a-lon2) (second a-lon2)) (sorted-lon2? (rest a-lon2))))))
- ; Sample expressions for sorted-lon2
(define E-LON2-VAL
 (or (< (length E-LON2) 2)
 (and (<= (first E-LON2) (second E-LON2)) (sorted-lon2? (rest E-LON2))))))

(define A-LON2-VAL
 (or (< (length A-LON2) 2)
 (and (<= (first A-LON2) (second A-LON2)) (sorted-lon2? (rest A-LON2))))))

(define SORTED-LON2-VAL
 (or (< (length SORTED-LON2) 2)
 (and (<= (first SORTED-LON2) (second SORTED-LON2))
 (sorted-lon2? (rest SORTED-LON2))))))

(define UNSORTED-LON2-VAL
 (or (< (length UNSORTED-LON2) 2)
 (and (<= (first UNSORTED-LON2)
 (second UNSORTED-LON2))
 (sorted-lon2? (rest UNSORTED-LON2))))))
- ; Tests using sample computations for sorted-LON2
(check-expect (sorted-lon2? E-LON2) E-LON2-VAL)
(check-expect (sorted-lon2? A-LON2) A-LON2-VAL)
(check-expect (sorted-lon2? SORTED-LON2) SORTED-LON2-VAL)
(check-expect (sorted-lon2? UNSORTED-LON2) UNSORTED-LON2-VAL)
;
; Tests using sample values for sorted-lon2
(check-expect (sorted-lon2? '(8 7 6 5)) #false)
(check-expect (sorted-lon2? '(5 6 7 8)) #true)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List ANDing

- Problems: 139–142, 145

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Mapping

- List mapping refers to the process of applying a function to every element of a list and returning a list of the results
- This type of operation is very common when solving problems involving lists

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Mapping

- List mapping refers to the process of applying a function to every element of a list and returning a list of the results
- This type of operation is very common when solving problems involving lists
- In Aliens Attack multiple aliens and multiple shots may be represented using, respectively, a list of aliens and a list of shots
- Every time the clock ticks all the aliens and all the shots must be moved
- How is this accomplished?

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Mapping

- Consider how to move every alien
- Imagine the world in Aliens Attack contains a list of aliens and a direction

List Processing

List Mapping

- Consider how to move every alien
- Imagine the world in Aliens Attack contains a list of aliens and a direction
- The function `move-alien` must be mapped over the given list to create the list of the moved aliens

List Processing

List Mapping

- Consider how to move every alien
- Imagine the world in Aliens Attack contains a list of aliens and a direction
- The function `move-alien` must be mapped over the given list to create the list of the moved aliens
- Think about how to process the list of aliens based on its structure

List Processing

List Mapping

- Consider how to move every alien
- Imagine the world in Aliens Attack contains a list of aliens and a direction
- The function `move-alien` must be mapped over the given list to create the list of the moved aliens
- Think about how to process the list of aliens based on its structure
- If the list of aliens is empty then there are no aliens to move and the list of moved aliens is empty

List Processing

List Mapping

- Consider how to move every alien
- Imagine the world in Aliens Attack contains a list of aliens and a direction
- The function `move-alien` must be mapped over the given list to create the list of the moved aliens
- Think about how to process the list of aliens based on its structure
- If the list of aliens is empty then there are no aliens to move and the list of moved aliens is empty
- Otherwise, the first alien is moved using the function `move-alien`
- The rest of the aliens are moved
- The list of moved aliens is created by consing these two values

List Processing

List Mapping

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ;; Sample expressions for move-loa

```
(define MELOA-VAL  E-LOA)
(define MILOA-VAL  (cons (move-alien (first INIT-LOA) 'left)
                           (move-loa (rest INIT-LOA) 'left)))
(define MILOA-VAL2 (cons (move-alien (first INIT-LOA) 'right)
                           (move-loa (rest INIT-LOA) 'right)))
(define MILOA-VAL3 (cons (move-alien (first INIT-LOA) 'down)
                           (move-loa (rest INIT-LOA) 'down)))
```

List Processing

List Mapping

- `;; loa dir → loa` Purpose: Move given loa in given dir
`(define (move-loa a-loa dir)`

- `;; Sample expressions for move-loa`
`(define MELOA-VAL E-LOA)`
`(define MILOA-VAL (cons (move-alien (first INIT-LOA) 'left)`
`(move-loa (rest INIT-LOA) 'left)))`
`(define MILOA-VAL2 (cons (move-alien (first INIT-LOA) 'right)`
`(move-loa (rest INIT-LOA) 'right)))`
`(define MILOA-VAL3 (cons (move-alien (first INIT-LOA) 'down)`
`(move-loa (rest INIT-LOA) 'down)))`

List Processing

List Mapping

- `;; loa dir → loa Purpose: Move given loa in given dir
(define (move-loa a-loa dir)`

- `;; Sample expressions for move-loa
(define MELOA-VAL E-LOA)
(define MILOA-VAL (cons (move-alien (first INIT-LOA) 'left)
 (move-loa (rest INIT-LOA) 'left)))
(define MILOA-VAL2 (cons (move-alien (first INIT-LOA) 'right)
 (move-loa (rest INIT-LOA) 'right)))
(define MILOA-VAL3 (cons (move-alien (first INIT-LOA) 'down)
 (move-loa (rest INIT-LOA) 'down)))`
- `;; Tests using sample computations for move-loa
(check-expect (move-loa E-LOA 'right) MELOA-VAL)
(check-expect (move-loa INIT-LOA 'left) MILOA-VAL)
(check-expect (move-loa INIT-LOA 'right) MILOA-VAL2)
(check-expect (move-loa INIT-LOA 'down) MILOA-VAL3)
;; Tests using sample values for move-loa
(check-expect
 (move-loa (cons (make-posn 1 1) (cons (make-posn 1 2) '())) 'right)
 (cons (make-posn 2 1) (cons (make-posn 2 2) '())))

(check-expect (move-loa (cons (make-posn 1 1) (cons (make-posn 1 2) '()))) 'left)
 (cons (make-posn 0 1) (cons (make-posn 0 2) '()))))
(check-expect (move-loa (cons (make-posn 1 1) (cons (make-posn 1 2) '()))) 'down)
 (cons (make-posn 1 2) (cons (make-posn 1 3) '()))))`

List Processing

List Mapping

- `;; loa dir → loa Purpose: Move given loa in given dir`
`(define (move-loa a-loa dir)`
- `(if (empty? a-loa)`
 `E-LOA`
 `(cons (move-alien (first a-loa) dir)`
 `(move-loa (rest a-loa) dir))))`
- `; Sample expressions for move-loa`
`(define MELOA-VAL E-LOA)`
`(define MILOA-VAL (cons (move-alien (first INIT-LOA) 'left)`
 `(move-loa (rest INIT-LOA) 'left))))`
`(define MILOA-VAL2 (cons (move-alien (first INIT-LOA) 'right)`
 `(move-loa (rest INIT-LOA) 'right))))`
`(define MILOA-VAL3 (cons (move-alien (first INIT-LOA) 'down)`
 `(move-loa (rest INIT-LOA) 'down))))`
- `; Tests using sample computations for move-loa`
`(check-expect (move-loa E-LOA 'right) MELOA-VAL)`
`(check-expect (move-loa INIT-LOA 'left) MILOA-VAL)`
`(check-expect (move-loa INIT-LOA 'right) MILOA-VAL2)`
`(check-expect (move-loa INIT-LOA 'down) MILOA-VAL3)`
`; Tests using sample values for move-loa`
`(check-expect`
 `(move-loa (cons (make-posn 1 1) (cons (make-posn 1 2) '())) 'right)`
 `(cons (make-posn 2 1) (cons (make-posn 2 2) '()))))`

`(check-expect (move-loa (cons (make-posn 1 1) (cons (make-posn 1 2) '()))) 'left)`
 `(cons (make-posn 0 1) (cons (make-posn 0 2) '()))))`
`(check-expect (move-loa (cons (make-posn 1 1) (cons (make-posn 1 2) '()))) 'down)`
 `(cons (make-posn 1 2) (cons (make-posn 1 3) '()))))`

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Mapping

- Similar for moving a list of shots
- Read the textbook

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Mapping

- A mapping function may take as input a (listof X) and return a (listof Y) if the signature of the mapped function is: $X \rightarrow Y$
- Consider computing the lengths in a (listof string)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Mapping

- A mapping function may take as input a (`listof X`) and return a (`listof Y`) if the signature of the mapped function is: $X \rightarrow Y$
- Consider computing the lengths in a (`listof string`)
- The length of a string is always a natural number
- The contract for a function that solves this problem is:
 $(\text{listof string}) \rightarrow (\text{listof natnum})$

List Processing

List Mapping

- ;; Sample instances of a (listof string)
(define E-LOSTR '())
(define NE-LOSTR '("Here's" "a" "sigh" "to" "those" "who" "love" "me."))
(define NE-LOSTR2 '("And" "a" "smile" "for" "to" "those" "who" "hate"))

List Processing

List Mapping

- ;; Sample instances of a (listof string)
(define E-LOSTR '())
(define NE-LOSTR '("Here's" "a" "sigh" "to" "those" "who" "love" "me."))
(define NE-LOSTR2 '("And" "a" "smile" "for" "to" "those" "who" "hate"))

- ;; Sample expressions for lengths-lostr
(define E-LOSTR-VAL '())
(define NE-LOSTR-VAL (cons (string-length (first NE-LOSTR))
 (lengths-lostr (rest NE-LOSTR))))
(define NE-LOSTR2-VAL (cons (string-length (first NE-LOSTR))
 (lengths-lostr (rest NE-LOSTR))))

List Processing

List Mapping

- ```
;; Sample instances of a (listof string)
(define E-LOSTR '())
(define NE-LOSTR ("Here's" "a" "sigh" "to" "those" "who" "love" "me."))
(define NE-LOSTR2 ('("And" "a" "smile" "for" "to" "those" "who" "hate"))
```
- ```
;; (listof string) → lon
;; Purpose: Return list of the string lengths from given (listof string)
(define (lengths-lostr a-lostr)
```
- ```
;; Sample expressions for lengths-lostr
(define E-LOSTR-VAL '())
(define NE-LOSTR-VAL (cons (string-length (first NE-LOSTR))
 (lengths-lostr (rest NE-LOSTR))))
(define NE-LOSTR2-VAL (cons (string-length (first NE-LOSTR))
 (lengths-lostr (rest NE-LOSTR))))
```

# List Processing

## List Mapping

- ```
;; Sample instances of a (listof string)
(define E-LOSTR  '())
(define NE-LOSTR  ("Here's" "a" "sigh" "to" "those" "who" "love" "me."))
(define NE-LOSTR2 ('(And" "a" "smile" "for" "to" "those" "who" "hate"))
```
- ```
(listof string) → lon
;; Purpose: Return list of the string lengths from given (listof string)
(define (lengths-lostr a-lostr)
```
- ```
;; Sample expressions for lengths-lostr
(define E-LOSTR-VAL '())
(define NE-LOSTR-VAL  (cons (string-length (first NE-LOSTR))
                           (lengths-lostr (rest NE-LOSTR))))
(define NE-LOSTR2-VAL (cons (string-length (first NE-LOSTR))
                           (lengths-lostr (rest NE-LOSTR))))
```
- ```
; Tests using sample computations for lengths-lostr
(check-expect (lengths-lostr E-LOSTR) E-LOSTR-VAL)
(check-expect (lengths-lostr NE-LOSTR) NE-LOSTR-VAL)

;; Tests using sample values for lengths-lostr
(check-expect (lengths-lostr
 (list "And" "whatever" "sky's" "above" "me"))
 (list 3 8 5 5 2))
(check-expect (lengths-lostr
 (list "Here's" "a" "heart" "for" "every" "fate"))
 (list 6 1 5 3 5 4))
```

# List Processing

## List Mapping

- ```
;; Sample instances of a (listof string)
(define E-LOSTR  '())
(define NE-LOSTR  ("Here's" "a" "sigh" "to" "those" "who" "love" "me."))
(define NE-LOSTR2 ('(And" "a" "smile" "for" "to" "those" "who" "hate"))
```
- ```
(listof string) → lon
;; Purpose: Return list of the string lengths from given (listof string)
(define (lengths-lostr a-lostr))
```
- ```
(if (empty? a-lostr)
      '()
```
- ```
; Sample expressions for lengths-lostr
(define E-LOSTR-VAL '())
(define NE-LOSTR-VAL (cons (string-length (first NE-LOSTR))
 (lengths-lostr (rest NE-LOSTR))))
(define NE-LOSTR2-VAL (cons (string-length (first NE-LOSTR))
 (lengths-lostr (rest NE-LOSTR))))
```
- ```
; Tests using sample computations for lengths-lostr
(check-expect (lengths-lostr E-LOSTR) E-LOSTR-VAL)
(check-expect (lengths-lostr NE-LOSTR) NE-LOSTR-VAL)
```



```
; Tests using sample values for lengths-lostr
(check-expect (lengths-lostr
                (list "And" "whatever" "sky's" "above" "me"))
              (list 3 8 5 5 2))
(check-expect (lengths-lostr
                (list "Here's" "a" "heart" "for" "every" "fate"))
              (list 6 1 5 3 5 4))
```

List Processing

List Mapping

- ```
;; Sample instances of a (listof string)
(define E-LOSTR '())
(define NE-LOSTR ("Here's" "a" "sigh" "to" "those" "who" "love" "me."))
(define NE-LOSTR2 ('("And" "a" "smile" "for" "to" "those" "who" "hate")))
```
- ```
;; (listof string) → lon
;; Purpose: Return list of the string lengths from given (listof string)
(define (lengths-lostr a-lostr))
```
- ```
(if (empty? a-lostr)
 '()
 (cons (string-length (first a-lostr)) (lengths-lostr (rest a-lostr))))
```
- ```
; Sample expressions for lengths-lostr
(define E-LOSTR-VAL '())
(define NE-LOSTR-VAL  (cons (string-length (first NE-LOSTR))
                            (lengths-lostr (rest NE-LOSTR))))
(define NE-LOSTR2-VAL (cons (string-length (first NE-LOSTR))
                            (lengths-lostr (rest NE-LOSTR))))
```
- ```
; Tests using sample computations for lengths-lostr
(check-expect (lengths-lostr E-LOSTR) E-LOSTR-VAL)
(check-expect (lengths-lostr NE-LOSTR) NE-LOSTR-VAL)

;; Tests using sample values for lengths-lostr
(check-expect (lengths-lostr
 (list "And" "whatever" "sky's" "above" "me"))
 (list 3 8 5 5 2))
(check-expect (lengths-lostr
 (list "Here's" "a" "heart" "for" "every" "fate"))
 (list 6 1 5 3 5 4))
```

**Part III:**  
**Compound**  
**Data of**  
**Arbitrary**  
**Size**

**Marco T.**  
**Morazán**

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## Homework

- Problems: 147–151

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- List filtering refers to the process of removing/filtering elements from a given list that do not satisfy a condition
- Alternatively, you may think of list filtering as returning/extracting a list of the elements that satisfy a given condition
- This is achieved by testing every element of the list as it is traversed

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- Multiple shots and multiple aliens means that aliens hit by a shot must be removed from the world's list of aliens
- Accomplished by traversing the `loa` and testing each alien

# List Processing

## List Filtering

- Multiple shots and multiple aliens means that aliens hit by a shot must be removed from the world's list of aliens
- Accomplished by traversing the `loa` and testing each alien
- If the first alien has been hit by any shot then it is not added to the resulting `loa`
- Otherwise the first alien is added to the resulting `loa`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- Multiple shots and multiple aliens means that aliens hit by a shot must be removed from the world's list of aliens
- Accomplished by traversing the `loa` and testing each alien
- If the first alien has been hit by any shot then it is not added to the resulting `loa`
- Otherwise the first alien is added to the resulting `loa`
- Problem analysis reveals that a conditional expression with three stanzas is needed

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- ;; Sample expressions for remove-hit-aliens

```
(define EMP-LOA-VAL '())
(define INIT-LOA-VAL (cons (first INIT-LOA)
 (remove-hit-aliens (rest INIT-LOA) INIT-LOS)))
(define INIT-LOA-VAL2 (remove-hit-aliens (rest INIT-LOA) LOS2))
```

# List Processing

## List Filtering

- ;; loa los → loa Purpose: Remove hit aliens  
`(define (remove-hit-aliens a-loa a-los)`

- ;; Sample expressions for remove-hit-aliens  
`(define EMP-LOA-VAL '())  
(define INIT-LOA-VAL (cons (first INIT-LOA)  
 (remove-hit-aliens (rest INIT-LOA) INIT-LOS)))  
(define INIT-LOA-VAL2 (remove-hit-aliens (rest INIT-LOA) LOS2))`

# List Processing

## List Filtering

- ;; loa los → loa Purpose: Remove hit aliens  
`(define (remove-hit-aliens a-loa a-los)`

- ;; Sample expressions for remove-hit-aliens  
`(define EMP-LOA-VAL '())  
(define INIT-LOA-VAL (cons (first INIT-LOA)  
 (remove-hit-aliens (rest INIT-LOA) INIT-LOS)))  
(define INIT-LOA-VAL2 (remove-hit-aliens (rest INIT-LOA) LOS2))`
- ;; Tests using sample computations for remove-hit-aliens  
`(check-expect (remove-hit-aliens E-LOA LOS2) EMP-LOA-VAL)  
(check-expect (remove-hit-aliens INIT-LOA INIT-LOS) INIT-LOA-VAL)  
(check-expect (remove-hit-aliens INIT-LOA LOS2) INIT-LOA-VAL2)  
(check-expect  
 (remove-hit-aliens  
 (cons (make-posn 1 1) (cons (make-posn 2 2)  
 (cons (make-posn 3 3) (cons (make-posn 4 4) '()))))  
 (cons (make-posn 3 3) (cons NO-SHOT (cons (make-posn 1 1) '()))))  
 (cons (make-posn 2 2) (cons (make-posn 4 4) '()))))  
  
(check-expect  
 (remove-hit-aliens (cons (make-posn 1 1) (cons (make-posn 2 2)  
 (cons (make-posn 3 3) '()))))  
 (cons (make-posn 7 4) (cons NO-SHOT (cons (make-posn 3 5) '()))))  
 (cons (make-posn 1 1) (cons (make-posn 2 2) (cons (make-posn 3 3) '())))))`

# List Processing

## List Filtering

- ;; loa los → loa Purpose: Remove hit aliens  
`(define (remove-hit-aliens a-loa a-los)`  
 `(cond [(empty? a-loa) '()]`  
 `[(hit-by-any-shot? (first a-loa) a-los)`  
 `(remove-hit-aliens (rest a-loa) a-los)]`  
 `[else (cons (first a-loa)`  
 `(remove-hit-aliens (rest a-loa) a-los))]))`
- ;; Sample expressions for remove-hit-aliens  
`(define EMP-LOA-VAL '())`  
`(define INIT-LOA-VAL (cons (first INIT-LOA)`  
 `(remove-hit-aliens (rest INIT-LOA) INIT-LOS)))`  
`(define INIT-LOA-VAL2 (remove-hit-aliens (rest INIT-LOA) LOS2))`
- ;; Tests using sample computations for remove-hit-aliens  
`(check-expect (remove-hit-aliens E-LOA LOS2) EMP-LOA-VAL)`  
`(check-expect (remove-hit-aliens INIT-LOA INIT-LOS) INIT-LOA-VAL)`  
`(check-expect (remove-hit-aliens INIT-LOA LOS2) INIT-LOA-VAL2)`  
`(check-expect`  
 `(remove-hit-aliens`  
 `(cons (make-posn 1 1) (cons (make-posn 2 2)`  
 `(cons (make-posn 3 3) (cons (make-posn 4 4) '()))))`  
 `(cons (make-posn 3 3) (cons NO-SHOT (cons (make-posn 1 1) '()))))`  
 `(cons (make-posn 2 2) (cons (make-posn 4 4) '()))))`
- (check-expect  
 (remove-hit-aliens (cons (make-posn 1 1) (cons (make-posn 2 2)  
 (cons (make-posn 3 3) '()))))  
 (cons (make-posn 7 4) (cons NO-SHOT (cons (make-posn 3 5) '()))))  
 (cons (make-posn 1 1) (cons (make-posn 2 2) (cons (make-posn 3 3) '()))))

**Part III:  
Compound  
Data of  
Arbitrary  
Size**

**Marco T.  
Morazán**

**Lists**

**List  
Processing**

**Natural  
Numbers**

**Interval  
Processing**

**Aliens Attack  
Version 4**

**Binary Trees**

**Mutually  
Recursive  
Data**

**Processing  
Multiple  
Inputs of  
Arbitrary  
Size**

# List Processing

## List Filtering

- How do we determine if the alien has been hit by any shot in a given los?

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- How do we determine if the alien has been hit by any shot in a given los?
- The los must be traversed to determine if hit? holds for any of the shots
- Determining if a predicate holds for any member of a list is a list oring function
- An and-expression is used to determine that the given list of shots is not empty and an or-expression is used to determine if the first shot has hit the alien or some other shot has hit the alien

# List Processing

## List Filtering

- ;; Sample expressions for hit-by-any-shot?  

```
(define HIT-LOS1-VAL (and (not (empty? LOS2))
 (or (hit? (first LOS2) ALIEN-8-0)
 (hit-by-any-shot? ALIEN-8-0
 (rest LOS2)))))

(define HIT-LOS2-VAL (and (not (empty? INIT-LOS))
 (or (hit? (first INIT-LOS) ALIEN-8-0)
 (hit-by-any-shot?
 ALIEN-8-0
 (rest INIT-LOS)))))
```

# List Processing

## List Filtering

- ```
;; alien los → Boolean
;; Purpose: To determine if the given alien is hit by any shot
;;           in the given los
(define (hit-by-any-shot? an-alien a-los)
```
- ```
;; Sample expressions for hit-by-any-shot?
(define HIT-LOS1-VAL (and (not (empty? LOS2))
 (or (hit? (first LOS2) ALIEN-8-0)
 (hit-by-any-shot? ALIEN-8-0
 (rest LOS2)))))

(define HIT-LOS2-VAL (and (not (empty? INIT-LOS))
 (or (hit? (first INIT-LOS) ALIEN-8-0)
 (hit-by-any-shot?
 ALIEN-8-0
 (rest INIT-LOS)))))
```

# List Processing

## List Filtering

- ```
;; alien los → Boolean
;; Purpose: To determine if the given alien is hit by any shot
;;           in the given los
(define (hit-by-any-shot? an-alien a-los)
```
- ```
;; Sample expressions for hit-by-any-shot?
(define HIT-LOS1-VAL (and (not (empty? LOS2))
 (or (hit? (first LOS2) ALIEN-8-0)
 (hit-by-any-shot? ALIEN-8-0
 (rest LOS2)))))

(define HIT-LOS2-VAL (and (not (empty? INIT-LOS))
 (or (hit? (first INIT-LOS) ALIEN-8-0)
 (hit-by-any-shot?
 ALIEN-8-0
 (rest INIT-LOS)))))
```
- ```
; Tests using sample computations for hit-by-any-shot?
(check-expect (hit-by-any-shot? ALIEN-8-0 LOS2) HIT-LOS1-VAL)
(check-expect (hit-by-any-shot? ALIEN-8-0 INIT-LOS) HIT-LOS2-VAL)

;; Tests using sample values for hit-by-any-shot?
(check-expect (hit-by-any-shot? (make-posn 8 3)
                                (list (make-posn 1 1) (make-posn 10 7)))
                  #false)
(check-expect (hit-by-any-shot? (make-posn 10 7)
                                (list (make-posn 1 1) (make-posn 10 7)))
                  #true)
```

List Processing

List Filtering

- ```
;; alien los → Boolean
;; Purpose: To determine if the given alien is hit by any shot
;; in the given los
(define (hit-by-any-shot? an-alien a-los)
```
- ```
(and (not (empty? a-los))
      (or (hit? (first a-los) an-alien)
          (hit-by-any-shot? an-alien (rest a-los)))))
```
- ```
; Sample expressions for hit-by-any-shot?
(define HIT-LOS1-VAL (and (not (empty? LOS2))
 (or (hit? (first LOS2) ALIEN-8-0)
 (hit-by-any-shot? ALIEN-8-0
 (rest LOS2)))))

(define HIT-LOS2-VAL (and (not (empty? INIT-LOS))
 (or (hit? (first INIT-LOS) ALIEN-8-0)
 (hit-by-any-shot?
 ALIEN-8-0
 (rest INIT-LOS)))))
```
- ```
; Tests using sample computations for hit-by-any-shot?
(check-expect (hit-by-any-shot? ALIEN-8-0 LOS2) HIT-LOS1-VAL)
(check-expect (hit-by-any-shot? ALIEN-8-0 INIT-LOS) HIT-LOS2-VAL)

;; Tests using sample values for hit-by-any-shot?
(check-expect (hit-by-any-shot? (make-posn 8 3)
                                (list (make-posn 1 1) (make-posn 10 7)))
              #false)
(check-expect (hit-by-any-shot? (make-posn 10 7)
                                (list (make-posn 1 1) (make-posn 10 7)))
              #true))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Filtering

- In Aliens Attack, should a single shot be able to hit multiple aliens or should it only be able to hit one alien?

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

List Filtering

- In Aliens Attack, should a single shot be able to hit multiple aliens or should it only be able to hit one alien?
- This is a design choice
- We adopt the second posture

List Processing

List Filtering

- In Aliens Attack, should a single shot be able to hit multiple aliens or should it only be able to hit one alien?
- This is a design choice
- We adopt the second posture
- The list of shots must be filtered to eliminate shots that have hit an alien
- Extract the shots that have not hit an alien

List Processing

List Filtering

- In Aliens Attack, should a single shot be able to hit multiple aliens or should it only be able to hit one alien?
- This is a design choice
- We adopt the second posture
- The list of shots must be filtered to eliminate shots that have hit an alien
- Extract the shots that have not hit an alien
- A posn shot that has the same coordinates as any alien must be filtered out
- In addition, all NO-SHOT will also be filtered out because such a shot no longer can affect the evolution of the game

List Processing

List Filtering

- In Aliens Attack, should a single shot be able to hit multiple aliens or should it only be able to hit one alien?
- This is a design choice
- We adopt the second posture
- The list of shots must be filtered to eliminate shots that have hit an alien
- Extract the shots that have not hit an alien
- A posn shot that has the same coordinates as any alien must be filtered out
- In addition, all NO-SHOT will also be filtered out because such a shot no longer can affect the evolution of the game
- To filter the shots we need as input the list of shots and the list of aliens

List Processing

List Filtering

- ```
;; Sample loa instances
(define LOA3 (list (make-posn 1 9) (make-posn 8 0)))
(define LOA4 (list (make-posn 1 9) (make-posn 8 5)))
;; Sample expressions for remove-shots
(define RM-INIT-LOS-VAL INIT-LOS)
(define RM-LOS2-VAL (remove-shots (rest LOS2) LOA3))
(define RM-LOS2-VAL2 (cons (make-posn 8 0)
 (remove-shots (rest LOS2) LOA4)))
```

# List Processing

## List Filtering

- `; los loa → los Purpose: Remove hit and NO-SHOTs from the los  
(define (remove-shots a-los a-loa)`

- `; Sample loa instances  
(define LOA3 (list (make-posn 1 9) (make-posn 8 0)))  
(define LOA4 (list (make-posn 1 9) (make-posn 8 5)))  
;; Sample expressions for remove-shots  
(define RM-INIT-LOS-VAL INIT-LOS)  
(define RM-LOS2-VAL (remove-shots (rest LOS2) LOA3))  
(define RM-LOS2-VAL2 (cons (make-posn 8 0)  
 (remove-shots (rest LOS2) LOA4)))`

# List Processing

## List Filtering

- `;; los loa → los Purpose: Remove hit and NO-SHOTs from the los  
(define (remove-shots a-los a-loa)`

- `;; Sample loa instances  
(define LOA3 (list (make-posn 1 9) (make-posn 8 0)))  
(define LOA4 (list (make-posn 1 9) (make-posn 8 5)))  
;; Sample expressions for remove-shots  
(define RM-INIT-LOS-VAL INIT-LOS)  
(define RM-LOS2-VAL (remove-shots (rest LOS2) LOA3))  
(define RM-LOS2-VAL2 (cons (make-posn 8 0)  
 (remove-shots (rest LOS2) LOA4)))`
- `;; Tests using sample computations for remove-shots  
(check-expect (remove-shots INIT-LOS INIT-LOA) RM-INIT-LOS-VAL)  
(check-expect (remove-shots LOS2 (list (make-posn 1 9) (make-posn 8 0)))  
 RM-LOS2-VAL)  
(check-expect (remove-shots LOS2 (list (make-posn 1 9) (make-posn 8 5)))  
 RM-LOS2-VAL2)  
;; Tests using sample values for remove-shots  
(check-expect (remove-shots (list (make-posn 2 9) (make-posn 10 10)) INIT-LOA)  
 (list (make-posn 2 9) (make-posn 10 10)))  
(check-expect (remove-shots (list (make-posn 8 2) (make-posn 13 1)) INIT-LOA)  
 '())`

# List Processing

## List Filtering

- `; ; los loa → los Purpose: Remove hit and NO-SHOTs from the los  
(define (remove-shots a-los a-loa)`
- `(cond [(empty? a-los) a-los]  
 [((or (eq? (first a-los) NO-SHOT)  
 (hit-any-alien? (first a-los) a-loa))  
 (remove-shots (rest a-los) a-loa)]  
 [else (cons (first a-los)  
 (remove-shots (rest a-los) a-loa))]))`
- `; ; Sample loa instances  
(define LOA3 (list (make-posn 1 9) (make-posn 8 0)))  
(define LOA4 (list (make-posn 1 9) (make-posn 8 5)))  
;; Sample expressions for remove-shots  
(define RM-INIT-LOS-VAL INIT-LOS)  
(define RM-LOS2-VAL (remove-shots (rest LOS2) LOA3))  
(define RM-LOS2-VAL2 (cons (make-posn 8 0)  
 (remove-shots (rest LOS2) LOA4)))`
- `; ; Tests using sample computations for remove-shots  
(check-expect (remove-shots INIT-LOS INIT-LOA) RM-INIT-LOS-VAL)  
(check-expect (remove-shots LOS2 (list (make-posn 1 9) (make-posn 8 0)))  
 RM-LOS2-VAL)  
(check-expect (remove-shots LOS2 (list (make-posn 1 9) (make-posn 8 5)))  
 RM-LOS2-VAL2)  
;; Tests using sample values for remove-shots  
(check-expect (remove-shots (list (make-posn 2 9) (make-posn 10 10)) INIT-LOA)  
 (list (make-posn 2 9) (make-posn 10 10)))  
(check-expect (remove-shots (list (make-posn 8 2) (make-posn 13 1)) INIT-LOA)  
 '())`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- To determine if a shot has hit an alien the given loa must be traversed

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Filtering

- To determine if a shot has hit an alien the given loa must be traversed
- A given shot has hit any alien if the given loa is not empty and either the shot has hit the first alien or the shot has hit any other alien
- This is a list ORing operation.

# List Processing

## List Filtering

- ;; Sample instance of a shot  
`(define SHOT5 (make-posn 11 1))  
;; Sample expressions for hit-any-alien?  
(define HIT-ANY-ALIENO  
 (and (not (empty? E-LOA)) (or (hit? NO-SHOT (first E-LOA))  
 (hit-any-alien? NO-SHOT (rest E-LOA))))))  
(define HIT-ANY-ALIEN1  
 (and (not (empty? E-LOA)) (or (hit? SHOT3 (first INIT-LOA))  
 (hit-any-alien? NO-SHOT (rest INIT-LOA))))))  
(define HIT-ANY-ALIEN2  
 (and (not (empty? INIT-LOA)) (or (hit? NO-SHOT (first INIT-LOA))  
 (hit-any-alien? NO-SHOT (rest INIT-LOA))))))  
(define HIT-ANY-ALIEN3  
 (and (not (empty? INIT-LOA)) (or (hit? SHOT5 (first INIT-LOA))  
 (hit-any-alien? SHOT5 (rest INIT-LOA))))))`

# List Processing

## List Filtering

- ;; shot loa → Boolean Purpose: Determine if shot hit any alien  
`(define (hit-any-alien? a-shot a-loa)`
- ;; Sample instance of a shot  
`(define SHOT5 (make-posn 11 1))`  
;; Sample expressions for hit-any-alien?  
`(define HIT-ANY-ALIEN0  
 (and (not (empty? E-LOA)) (or (hit? NO-SHOT (first E-LOA))  
 (hit-any-alien? NO-SHOT (rest E-LOA)))))`  
`(define HIT-ANY-ALIEN1  
 (and (not (empty? E-LOA)) (or (hit? SHOT3 (first INIT-LOA))  
 (hit-any-alien? NO-SHOT (rest INIT-LOA)))))`  
`(define HIT-ANY-ALIEN2  
 (and (not (empty? INIT-LOA)) (or (hit? NO-SHOT (first INIT-LOA))  
 (hit-any-alien? NO-SHOT (rest INIT-LOA)))))`  
`(define HIT-ANY-ALIEN3  
 (and (not (empty? INIT-LOA)) (or (hit? SHOT5 (first INIT-LOA))  
 (hit-any-alien? SHOT5 (rest INIT-LOA)))))`

# List Processing

## List Filtering

- ```
;; shot loa → Boolean Purpose: Determine if shot hit any alien
(define (hit-any-alien? a-shot a-loa)
```
- ```
;; Sample instance of a shot
(define SHOT5 (make-posn 11 1))
;; Sample expressions for hit-any-alien?
(define HIT-ANY-ALIEN0
 (and (not (empty? E-LOA)) (or (hit? NO-SHOT (first E-LOA))
 (hit-any-alien? NO-SHOT (rest E-LOA)))))
(define HIT-ANY-ALIEN1
 (and (not (empty? E-LOA)) (or (hit? SHOT3 (first INIT-LOA))
 (hit-any-alien? NO-SHOT (rest INIT-LOA)))))
(define HIT-ANY-ALIEN2
 (and (not (empty? INIT-LOA)) (or (hit? NO-SHOT (first INIT-LOA))
 (hit-any-alien? NO-SHOT (rest INIT-LOA))))
(define HIT-ANY-ALIEN3
 (and (not (empty? INIT-LOA)) (or (hit? SHOT5 (first INIT-LOA))
 (hit-any-alien? SHOT5 (rest INIT-LOA))))
```
- ```
; Tests using sample computations for hit-any-alien?
(check-expect (hit-any-alien? NO-SHOT E-LOA) HIT-ANY-ALIEN0)
(check-expect (hit-any-alien? SHOT3 E-LOA) HIT-ANY-ALIEN1)
(check-expect (hit-any-alien? NO-SHOT INIT-LOA) HIT-ANY-ALIEN2)
(check-expect (hit-any-alien? SHOT5 INIT-LOA) HIT-ANY-ALIEN3)
;; Tests using sample values for hit-any-alien?
(check-expect
  (hit-any-alien? (make-posn 9 3) (list (make-posn 6 2) (make-posn 9 3) (make-posn 9 11)))
  #true)
(check-expect
  (hit-any-alien? (make-posn 11 3) (list (make-posn 6 2)(make-posn 9 3) (make-posn 9 11)))
  #false)
```

List Processing

List Filtering

- ;; shot loa → Boolean Purpose: Determine if shot hit any alien

```
(define (hit-any-alien? a-shot a-loa)
  (and (not (empty? a-loa)) (or (hit? a-shot (first a-loa))
    (hit-any-alien? a-shot (rest a-loa))))))
```
- ;; Sample instance of a shot

```
(define SHOT5 (make-posn 11 1))
;; Sample expressions for hit-any-alien?
(define HIT-ANY-ALIEN0
  (and (not (empty? E-LOA)) (or (hit? NO-SHOT (first E-LOA))
    (hit-any-alien? NO-SHOT (rest E-LOA))))))
```
- ```
(define HIT-ANY-ALIEN1
 (and (not (empty? E-LOA)) (or (hit? SHOT3 (first INIT-LOA))
 (hit-any-alien? NO-SHOT (rest INIT-LOA))))))
```
- ```
(define HIT-ANY-ALIEN2
  (and (not (empty? INIT-LOA)) (or (hit? NO-SHOT (first INIT-LOA))
    (hit-any-alien? NO-SHOT (rest INIT-LOA))))))
```
- ```
(define HIT-ANY-ALIEN3
 (and (not (empty? INIT-LOA)) (or (hit? SHOT5 (first INIT-LOA))
 (hit-any-alien? SHOT5 (rest INIT-LOA))))))
```
- ;; Tests using sample computations for hit-any-alien?  

```
(check-expect (hit-any-alien? NO-SHOT E-LOA) HIT-ANY-ALIEN0)
(check-expect (hit-any-alien? SHOT3 E-LOA) HIT-ANY-ALIEN1)
(check-expect (hit-any-alien? NO-SHOT INIT-LOA) HIT-ANY-ALIEN2)
(check-expect (hit-any-alien? SHOT5 INIT-LOA) HIT-ANY-ALIEN3)
;; Tests using sample values for hit-any-alien?
(check-expect
 (hit-any-alien? (make-posn 9 3) (list (make-posn 6 2) (make-posn 9 3) (make-posn 9 11)))
 #true)
(check-expect
 (hit-any-alien? (make-posn 11 3) (list (make-posn 6 2)(make-posn 9 3) (make-posn 9 11)))
 #false)
```

**Part III:**  
**Compound**  
**Data of**  
**Arbitrary**  
**Size**

**Marco T.**  
**Morazán**

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## Homework

- Problems: 153–156, 159, 161

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Sorting

- The solution to many problems requires sorting a list
- In order for a list to be sortable it must contain numerical or ordinal data
- Any two elements are comparable to determine which goes first and which goes second
- Need a predicate that determines which of two elements goes first

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Sorting

- Sorting a list of numbers in non-decreasing order

# List Processing

## List Sorting

- Sorting a list of numbers in non-decreasing order
- '(87 65 90 21) → '(21 65 87 90)
- How is this accomplished?

# List Processing

## List Sorting

- Sorting a list of numbers in non-decreasing order
- '(87 65 90 21) → '(21 65 87 90)
- How is this accomplished?
- Think about the varieties of a list of numbers
- If the given lon is empty then the sorted lon is also empty.

# List Processing

## List Sorting

- Sorting a list of numbers in non-decreasing order
- '(87 65 90 21) → '(21 65 87 90)
- How is this accomplished?
- Think about the varieties of a list of numbers
- If the given lon is empty then the sorted lon is also empty.
- The template for a function on an lon tells us to process the rest of the list recursively:

(sort-lon (rest a-lon))

- This expression ought to evaluate to a sorted list containing all the elements of a-lon except a-lon's first element
- We must combine a-lon's first element with a sorted list
- How is this done?

# List Processing

## List Sorting

- Sorting a list of numbers in non-decreasing order
- '(87 65 90 21) → '(21 65 87 90)
- How is this accomplished?
- Think about the varieties of a list of numbers
- If the given lon is empty then the sorted lon is also empty.
- The template for a function on an lon tells us to process the rest of the list recursively:  

```
(sort-lon (rest a-lon))
```
- This expression ought to evaluate to a sorted list containing all the elements of a-lon except a-lon's first element
- We must combine a-lon's first element with a sorted list
- How is this done?
- The sorted list must be traversed to find the right place to insert a-lon's first element
- Inserting a number into a sorted list of numbers is a different problem from sorting a list of numbers meaning that an auxiliary function is needed

# List Processing

## List Sorting

- ; Sample instances of a lon  
`(define E-LON '())  
(define SORTED-LON '(17 18 29 37 41 52))  
(define UNSORTED-LON '(89 21 1 77 23))`

# List Processing

## List Sorting

- ;; Sample instances of a lon

```
(define E-LON '())
(define SORTED-LON '(17 18 29 37 41 52))
(define UNSORTED-LON '(89 21 1 77 23))
```

- ;; Sample expressions for sort-lon

```
(define E-LON-VAL '())
(define SORTED-LON-VAL (insert (first SORTED-LON)
 (sort-lon (rest SORTED-LON))))
(define UNSORTED-LON-VAL (insert (first UNSORTED-LON)
 (sort-lon (rest UNSORTED-LON))))
```

# List Processing

## List Sorting

- ;; Sample instances of a lon  

```
(define E-LON '())
(define SORTED-LON '(17 18 29 37 41 52))
(define UNSORTED-LON '(89 21 1 77 23))
```
- ;; sort-lon: lon '() lon Purpose: Sort lon in non-decreasing order  

```
(define (sort-lon a-lon)
```
- ;; Sample expressions for sort-lon  

```
(define E-LON-VAL '())
(define SORTED-LON-VAL (insert (first SORTED-LON)
 (sort-lon (rest SORTED-LON))))
(define UNSORTED-LON-VAL (insert (first UNSORTED-LON)
 (sort-lon (rest UNSORTED-LON))))
```

# List Processing

## List Sorting

- ;; Sample instances of a lon  
`(define E-LON '())  
(define SORTED-LON '(17 18 29 37 41 52))  
(define UNSORTED-LON '(89 21 1 77 23))`
- ;; sort-lon: lon '() lon Purpose: Sort lon in non-decreasing order  
`(define (sort-lon a-lon)`  
  
  
- ;; Sample expressions for sort-lon  
`(define E-LON-VAL '())  
(define SORTED-LON-VAL (insert (first SORTED-LON)  
 (sort-lon (rest SORTED-LON))))  
(define UNSORTED-LON-VAL (insert (first UNSORTED-LON)  
 (sort-lon (rest UNSORTED-LON))))`
- ;; Tests using sample computations for sort-lon  
`(check-expect (sort-lon E-LON) E-LON-VAL)  
(check-expect (sort-lon SORTED-LON) SORTED-LON-VAL)  
(check-expect (sort-lon UNSORTED-LON) UNSORTED-LON-VAL)  
;; Tests using sample values for sort-lon  
(check-expect (sort-lon (list 5 4 3 2 1)) (list 1 2 3 4 5))  
(check-expect (sort-lon (list 63 12 76 99 0)) (list 0 12 63 76 99))`

# List Processing

## List Sorting

- ;; Sample instances of a lon  
`(define E-LON '())  
(define SORTED-LON '(17 18 29 37 41 52))  
(define UNSORTED-LON '(89 21 1 77 23))`
- ;; sort-lon: lon '() lon Purpose: Sort lon in non-decreasing order  
`(define (sort-lon a-lon)  
 (cond [(empty? a-lon) '()]  
 [else (insert (first a-lon) (sort-lon (rest a-lon))))]))`
- ;; Sample expressions for sort-lon  
`(define E-LON-VAL '())  
(define SORTED-LON-VAL (insert (first SORTED-LON)  
 (sort-lon (rest SORTED-LON))))  
(define UNSORTED-LON-VAL (insert (first UNSORTED-LON)  
 (sort-lon (rest UNSORTED-LON))))`
- ;; Tests using sample computations for sort-lon  
`(check-expect (sort-lon E-LON) E-LON-VAL)  
(check-expect (sort-lon SORTED-LON) SORTED-LON-VAL)  
(check-expect (sort-lon UNSORTED-LON) UNSORTED-LON-VAL)  
;; Tests using sample values for sort-lon  
(check-expect (sort-lon (list 5 4 3 2 1)) (list 1 2 3 4 5))  
(check-expect (sort-lon (list 63 12 76 99 0)) (list 0 12 63 76 99))`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Sorting

- For `insert` we have a number and a sorted list

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Sorting

- For `insert` we have a number and a sorted list
- If the given sorted list is empty then the resulting list must only contain the given number

# List Processing

## List Sorting

- For `insert` we have a number and a sorted list
- If the given sorted list is empty then the resulting list must only contain the given number
- Otherwise, determine if the given number is less than or equal to the first number
- If so:

9 and '(10 43 88 100) → (cons 9 '(10 43 88 100))

# List Processing

## List Sorting

- For `insert` we have a number and a sorted list
- If the given sorted list is empty then the resulting list must only contain the given number
- Otherwise, determine if the given number is less than or equal to the first number
- If so:  
`9 and '(10 43 88 100) → (cons 9 '(10 43 88 100))`
- Given number is greater than the first number:  
`53 and '(10 43 88 100) → (cons 10 (insert 53 '(43 88 100)))`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# List Processing

## List Sorting

- (define SORTED-LON2 '(31 87 95 102))  
(define INSERT-NUM1 20) (define INSERT-NUM2 90)

Lists

## List Processing

## Natural Numbers

# Interval Processing

# Aliens Attack

## Version 4

## Binary Trees

# Mutually Recursive Data

## Processing Multiple Inputs of Arbitrary Size

# List Processing

## List Sorting

- (define SORTED-LONG '(31 87 95 102))  
(define INSERT-NUM1 20) (define INSERT-NUM2 90)

# List Processing

## List Sorting

# List Processing

## List Sorting

# List Processing

## List Sorting

- ```

• (define SORTED-LON2 '(31 87 95 102))
  (define INSERT-NUM1 20) (define INSERT-NUM2 90)
• ;; insert: number lon → lon Purpose: Insert number in sorted lon
  ;; ASSUMPTION: Given lon is sorted in nondecreasing order
  (define (insert a-num a-lon)
    (cond [(empty? a-lon) (list a-num)]
          [(<= a-num (first a-lon)) (cons a-num a-lon)]
          [else (cons (first a-lon) (insert a-num (rest a-lon)))])))
• ;; Sample expressions for insert
  (define ELON-VAL (list INSERT-NUM1))
  (define SORTEDLON-VAL (cons INSERT-NUM1 SORTED-LON2))
  (define UNSORTEDLON-VAL (cons (first SORTED-LON2)
                                 (insert INSERT-NUM2
                                       (rest SORTED-LON2))))
• ;; Tests using sample computations for insert
  (check-expect (insert INSERT-NUM1 E-LON) ELON-VAL)
  (check-expect (insert INSERT-NUM1 SORTED-LON2) SORTEDLON-VAL)
  (check-expect (cons (first SORTED-LON2)(insert INSERT-NUM2
                                                 (rest SORTED-LON2)))
                  UNSORTEDLON-VAL)
;; Tests using sample values for insert
  (check-expect (insert 3 '(1 2 4 5)) '(1 2 3 4 5))
  (check-expect (insert 101 '(-7 -5 0 2 232)) '(-7 -5 0 2 101 232))

```

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

List Processing

Homework

- Problems: 163, 165

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Natural Numbers

- In a Mathematics textbook you may see the following definition for the natural numbers:

The natural numbers are 0, 1, 2, ...
- You probably interpret the ... as meaning all the way to infinity
- Can you solve a problem if the given data is a natural number? How do you process a natural number?

Natural Numbers

- In a Mathematics textbook you may see the following definition for the natural numbers:

The natural numbers are 0, 1, 2, ...

- You probably interpret the ... as meaning all the way to infinity
- Can you solve a problem if the given data is a natural number? How do you process a natural number?
- How you compute $n!$?
- In a mathematics textbook you may find the following:

$$n! = 1 * 2 * * 3 * \dots * n-1 * n$$

- Do you understand how to compute $n!$?

Natural Numbers

- In a Mathematics textbook you may see the following definition for the natural numbers:

The natural numbers are 0, 1, 2, ...

- You probably interpret the ... as meaning all the way to infinity
- Can you solve a problem if the given data is a natural number? How do you process a natural number?
- How you compute $n!$?
- In a mathematics textbook you may find the following:

$$n! = 1 * 2 * * 3 * \dots * n-1 * n$$

- Do you understand how to compute $n!$?
- The problem is that we do not have a proper data definition for a natural number
- We do not know how to program ...

Natural Numbers

- A natural number may be small like 0

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Natural Numbers

- A natural number may be small like 0
- It may be large like 10000

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Natural Numbers

- A natural number may be small like 0
- It may be large like 10000
- It may be of medium size like 1587

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Natural Numbers

- A natural number may be small like 0
- It may be large like 10000
- It may be of medium size like 1587
- It may be super large like 100000000
- What is all this telling us?

Natural Numbers

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- A natural number may be small like 0
- It may be large like 10000
- It may be of medium size like 1587
- It may be super large like 100000000
- What is all this telling us?
- A natural number is data of arbitrary size
- What do we need to design functions that process a natural number?

Natural Numbers

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- A natural number may be small like 0
- It may be large like 10000
- It may be of medium size like 1587
- It may be super large like 100000000
- What is all this telling us?
- A natural number is data of arbitrary size
- What do we need to design functions that process a natural number?
- We need a recursive data definition and a function template.

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?
- Clearly, it is 0

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?
- Clearly, it is 0
- How can other natural numbers be created?

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?
- Clearly, it is 0
- How can other natural numbers be created?
- Let's start with 0: can we create the next natural number, 1, using 0?
- We increment 0 by 1
- Can we build, 2, the next natural number?

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?
- Clearly, it is 0
- How can other natural numbers be created?
- Let's start with 0: can we create the next natural number, 1, using 0?
- We increment 0 by 1
- Can we build, 2, the next natural number?
- Increment the natural number 1 by 1

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?
- Clearly, it is 0
- How can other natural numbers be created?
- Let's start with 0: can we create the next natural number, 1, using 0?
- We increment 0 by 1
- Can we build, 2, the next natural number?
- Increment the natural number 1 by 1
- There is a pattern here:

A natural number (`natnum`) is either:

1. 0
2. (`add1 natnum`)

- We can use `add1` to create a new `natnum`
- What is the selector function to extract the `natnum` used to build a given non-zero `natnum`?

Natural Numbers

Data Definition for a Natural Number

- What is the smallest natural number?
- Clearly, it is 0
- How can other natural numbers be created?
- Let's start with 0: can we create the next natural number, 1, using 0?
- We increment 0 by 1
- Can we build, 2, the next natural number?
- Increment the natural number 1 by 1
- There is a pattern here:

A natural number (`natnum`) is either:

1. 0
2. (`add1 natnum`)

- We can use `add1` to create a new `natnum`
- What is the selector function to extract the `natnum` used to build a given non-zero `natnum`?
- The selector function is `sub1`

Natural Numbers

Data Definition for a Natural Number

- #| ; Sample instances of natnum
(define ZERO 0)
(define NATNUMA ...) ...

;; natnum ... → ...
;; Purpose: ...
(define (f-on-natnum a-natnum ...)
(if (= a-natnum 0)
...
(...a-natnum...(f-on-natnum (sub1 a-natnum) ...))))

;; Sample expressions for f-on-natnum
(define ZERO-VAL ...)
(define NATNUMA-VAL ...) ...

;; Tests using sample computations for f-on-natnum
(check-expect (f-on-natnum ZERO ...) ZERO-VAL)
(check-expect (f-on-natnum NATNUMA ...) NATNUMA-VAL) ...

;; Tests using sample values for f-on-natnum
(check-expect (f-on-natnum ...) ...)
(check-expect (f-on-natnum ...) ...) ...

Natural Numbers

Computing Factorial

- Design a function to compute $n!$

- ```
;; Sample instances of natnum
(define ZERO 0)
(define TEN 10)
(define FIFTY 50)
```

# Natural Numbers

## Computing Factorial

- Design a function to compute  $n!$
- ```
;; Sample instances of natnum
(define ZERO 0)
(define TEN 10)
(define FIFTY 50)
```
- Reason statically about the structure of a natnum
- What is the factorial of 0?

Natural Numbers

Computing Factorial

- Design a function to compute $n!$
- ```
;; Sample instances of natnum
(define ZERO 0)
(define TEN 10)
(define FIFTY 50)
```
- Reason statically about the structure of a natnum
- What is the factorial of 0?
- It is not clear what the value of  $0!$  is
- If the given natnum,  $n$ , is not 0 the template suggests combining the given number and the factorial of  $n - 1$

# Natural Numbers

## Computing Factorial

- Design a function to compute  $n!$

- ```
;; Sample instances of natnum
(define ZERO 0)
(define TEN 10)
(define FIFTY 50)
```

- Reason statically about the structure of a natnum
- What is the factorial of 0?
- It is not clear what the value of $0!$ is
- If the given natnum, n , is not 0 the template suggests combining the given number and the factorial of $n - 1$
- $3! = 1 * 2 * 3$

$$4! = 1 * 2 * 3 * 4$$

Natural Numbers

Computing Factorial

- Design a function to compute $n!$

- ```
;; Sample instances of natnum
(define ZERO 0)
(define TEN 10)
(define FIFTY 50)
```

- Reason statically about the structure of a natnum
- What is the factorial of 0?
- It is not clear what the value of  $0!$  is
- If the given natnum,  $n$ , is not 0 the template suggests combining the given number and the factorial of  $n - 1$
- $3! = 1 * 2 * 3$

$$4! = 1 * 2 * 3 * 4$$

- $n$  and the value of  $(\text{sub1 } n)!$  must be multiplied
- A value must be returned when  $n$  is 0:

$$4! = 4 * 3 * 2 * 1 * ??$$

# Natural Numbers

## Computing Factorial

- Design a function to compute  $n!$
- ```
;; Sample instances of natnum
(define ZERO 0)
(define TEN 10)
(define FIFTY 50)
```
- Reason statically about the structure of a natnum
- What is the factorial of 0?
- It is not clear what the value of $0!$ is
- If the given natnum, n , is not 0 the template suggests combining the given number and the factorial of $n - 1$
- $3! = 1 * 2 * 3$

- $4! = 1 * 2 * 3 * 4$
- n and the value of $(\text{sub1 } n)!$ must be multiplied
- A value must be returned when n is 0:
- $4! = 4 * 3 * 2 * 1 * ??$
- Becomes clear that $0!$ is 1
- Any function that computes the factorial of a natnum must return a natnum.

Natural Numbers

Computing Factorial

- ```
;; Sample expressions for factorial
(define ZERO-VAL 1)
(define TEN-VAL (* 10 (factorial (sub1 10))))
(define FIFTY-VAL (* 50 (factorial (sub1 50))))
```

# Natural Numbers

## Computing Factorial

- ;; natnum → natnum  
;; Purpose: Compute the factorial of the given natnum  
(define (factorial a-natnum))
- ;; Sample expressions for factorial  
(define ZERO-VAL 1)  
(define TEN-VAL (\* 10 (factorial (sub1 10))))  
(define FIFTY-VAL (\* 50 (factorial (sub1 50))))

# Natural Numbers

## Computing Factorial

- ;; natnum → natnum  
;; Purpose: Compute the factorial of the given natnum  
(define (factorial a-natnum)
- ;; Sample expressions for factorial  
(define ZERO-VAL 1)  
(define TEN-VAL (\* 10 (factorial (sub1 10))))  
(define FIFTY-VAL (\* 50 (factorial (sub1 50))))
- ;; Tests using sample computations for factorial  
(check-expect (factorial ZERO) ZERO-VAL)  
(check-expect (factorial TEN) TEN-VAL)  
(check-expect (factorial FIFTY) FIFTY-VAL)
- ;; Tests using sample values for f-on-natnum  
(check-expect (factorial 3) 6)  
(check-expect (factorial 5) 120)

# Natural Numbers

## Computing Factorial

- ```
;; natnum → natnum
;; Purpose: Compute the factorial of the given natnum
(define (factorial a-natnum)

  • (if (= a-natnum 0)
        1
        (* a-natnum (factorial (sub1 a-natnum)))))

  • ;; Sample expressions for factorial
    (define ZERO-VAL 1)
    (define TEN-VAL   (* 10 (factorial (sub1 10))))
    (define FIFTY-VAL (* 50 (factorial (sub1 50)))))

  • ;; Tests using sample computations for factorial
    (check-expect (factorial ZERO) ZERO-VAL)
    (check-expect (factorial TEN)   TEN-VAL)
    (check-expect (factorial FIFTY) FIFTY-VAL)

    •;; Tests using sample values for f-on-natnum
      (check-expect (factorial 3) 6)
      (check-expect (factorial 5) 120)
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Natural Numbers

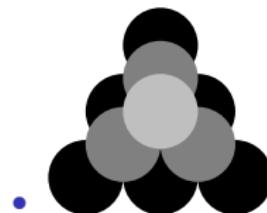
Homework

- Problems: 166–168

Natural Numbers

Computing Tetrahedral Numbers

- A tetrahedral is a triangular pyramid
- At each layer of a tetrahedral we find a set of objects, all the same, that form an equilateral triangle

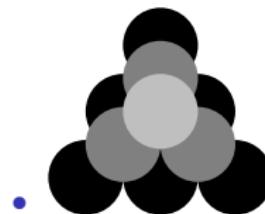


- The tetrahedral is of height three and is built using 10 disks: 6 for the bottom most layer, 3 for the middle layer, and 1 for the top layer

Natural Numbers

Computing Tetrahedral Numbers

- A tetrahedral is a triangular pyramid
- At each layer of a tetrahedral we find a set of objects, all the same, that form an equilateral triangle



- The tetrahedral is of height three and is built using 10 disks: 6 for the bottom most layer, 3 for the middle layer, and 1 for the top layer
- The numbers 1, 4, and 10 are the first three numbers
- The tetrahedral number in position n of the sequence is the number of objects needed to build a tetrahedral of height n

Natural Numbers

Computing Tetrahedral Numbers

- Imagine that you wish to build a tetrahedral of disks (all the same) of height 20
- How many disks do you need? (What is the 20th tetrahedral number?)

Natural Numbers

Computing Tetrahedral Numbers

- Imagine that you wish to build a tetrahedral of disks (all the same) of height 20
- How many disks do you need? (What is the 20th tetrahedral number?)
- We need to process a `natnum`
- Consider the structure of a natural number
- How many disks are needed to build a tetrahedral of height 0?

Natural Numbers

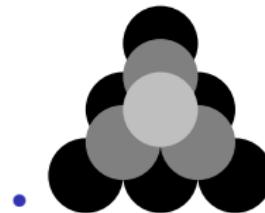
Computing Tetrahedral Numbers

- Imagine that you wish to build a tetrahedral of disks (all the same) of height 20
- How many disks do you need? (What is the 20th tetrahedral number?)
- We need to process a `natnum`
- Consider the structure of a natural number
- How many disks are needed to build a tetrahedral of height 0?
- If the height is 0 it means that it contains no disks and the 0th tetrahedral number is 0
- How many disks are needed if the height is not 0?

Natural Numbers

Computing Tetrahedral Numbers

- Imagine that you wish to build a tetrahedral of disks (all the same) of height 20
- How many disks do you need? (What is the 20th tetrahedral number?)
- We need to process a `natnum`
- Consider the structure of a natural number
- How many disks are needed to build a tetrahedral of height 0?
- If the height is 0 it means that it contains no disks and the 0th tetrahedral number is 0
- How many disks are needed if the height is not 0?
- Use a divide and conquer: the bottommost layer and all the other layers

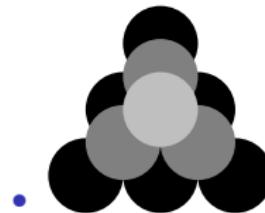


- Bottom layer: 6 and other layers: 4
- Total: $6 + 4 = 10$

Natural Numbers

Computing Tetrahedral Numbers

- Imagine that you wish to build a tetrahedral of disks (all the same) of height 20
- How many disks do you need? (What is the 20th tetrahedral number?)
- We need to process a `natnum`
- Consider the structure of a natural number
- How many disks are needed to build a tetrahedral of height 0?
- If the height is 0 it means that it contains no disks and the 0th tetrahedral number is 0
- How many disks are needed if the height is not 0?
- Use a divide and conquer: the bottommost layer and all the other layers

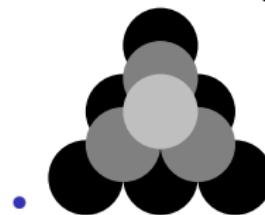


- Bottom layer: 6 and other layers: 4
- Total: $6 + 4 = 10$
- Bottom layer: Triangle of height 3
- Top layers: Tetrahedral of height 2

Natural Numbers

Computing Tetrahedral Numbers

- Imagine that you wish to build a tetrahedral of disks (all the same) of height 20
- How many disks do you need? (What is the 20th tetrahedral number?)
- We need to process a `natnum`
- Consider the structure of a natural number
- How many disks are needed to build a tetrahedral of height 0?
- If the height is 0 it means that it contains no disks and the 0th tetrahedral number is 0
- How many disks are needed if the height is not 0?
- Use a divide and conquer: the bottommost layer and all the other layers



- Bottom layer: 6 and other layers: 4
- Total: $6 + 4 = 10$
- Bottom layer: Triangle of height 3
- Top layers: Tetrahedral of height 2
- The number of disks needed to build a triangle is a triangular number
- $\text{tetra}(h) = \text{triangular}(h) + \text{tetra}(h-1)$

Natural Numbers

Computing Tetrahedral Numbers

- ;; Sample instances of natnum
(define ZERO 0)
(define NINE 9)
(define SIXTY 60)

Natural Numbers

Computing Tetrahedral Numbers

- ;; Sample instances of natnum
(define ZERO 0)
(define NINE 9)
(define SIXTY 60)

- ;; Sample expressions for nth-tetra
(define ZERO-VAL 0)
(define NINE-VAL (+ (nth-tri NINE) (nth-tetra (sub1 NINE))))
(define SIXTY-VAL (+ (nth-tri SIXTY) (nth-tetra (sub1 SIXTY)))))

Natural Numbers

Computing Tetrahedral Numbers

- ```
;; Sample instances of natnum
(define ZERO 0)
(define NINE 9)
(define SIXTY 60)
```
- ```
;; natnum → natnum
;; Purpose: Compute the nth tetrahedral number
(define (nth-tetra a-natnum)
```
- ```
;; Sample expressions for nth-tetra
(define ZERO-VAL 0)
(define NINE-VAL (+ (nth-tri NINE) (nth-tetra (sub1 NINE))))
(define SIXTY-VAL (+ (nth-tri SIXTY) (nth-tetra (sub1 SIXTY))))
```

# Natural Numbers

## Computing Tetrahedral Numbers

- ```
;; Sample instances of natnum
(define ZERO 0)
(define NINE 9)
(define SIXTY 60)
```
- ```
;; natnum → natnum
;; Purpose: Compute the n^{th} tetrahedral number
(define (nth-tetra a-natnum)
```
- ```
;; Sample expressions for nth-tetra
(define ZERO-VAL 0)
(define NINE-VAL (+ (nth-tri NINE) (nth-tetra (sub1 NINE))))
(define SIXTY-VAL (+ (nth-tri SIXTY) (nth-tetra (sub1 SIXTY))))
```
- ```
;; Tests using sample computations for nth-tetra
(check-expect (nth-tetra ZERO) ZERO-VAL)
(check-expect (nth-tetra NINE) NINE-VAL)
```
- ```
;; Tests using sample values for nth-tetra
(check-expect (nth-tetra 3) 10)
(check-expect (nth-tetra 2) 4)
```

Natural Numbers

Computing Tetrahedral Numbers

- ```
;; Sample instances of natnum
(define ZERO 0)
(define NINE 9)
(define SIXTY 60)
```
- ```
;; natnum → natnum
;; Purpose: Compute the  $n^{\text{th}}$  tetrahedral number
(define (nth-tetra a-natnum)
```
- ```
(if (= a-natnum 0)
 0
 (+ (nth-tri a-natnum) (nth-tetra (sub1 a-natnum)))))
```
- ```
;; Sample expressions for nth-tetra
(define ZERO-VAL 0)
(define NINE-VAL (+ (nth-tri NINE) (nth-tetra (sub1 NINE))))
(define SIXTY-VAL (+ (nth-tri SIXTY) (nth-tetra (sub1 SIXTY))))
```
- ```
;; Tests using sample computations for nth-tetra
(check-expect (nth-tetra ZERO) ZERO-VAL)
(check-expect (nth-tetra NINE) NINE-VAL)

;; Tests using sample values for nth-tetra
(check-expect (nth-tetra 3) 10)
(check-expect (nth-tetra 2) 4)
```

# Natural Numbers

## Computing Tetrahedral Numbers

- We now proceed to `nth-tri`.
- For every natural number there is a different disk of triangles of different height

(a) Height 1.



(b) Height 2.



(c) Height 3.



- Given that the height is a `natnum`, reason about the structure of a `natnum`
- How many disks are used for a triangle of height 0?

# Natural Numbers

## Computing Tetrahedral Numbers

- We now proceed to `nth-tri`.
- For every natural number there is a different disk of triangles of different height

(a) Height 1.



(b) Height 2.



(c) Height 3.



- Given that the height is a `natnum`, reason about the structure of a `natnum`
- How many disks are used for a triangle of height 0?
- Height zero means that 0 disks are used
- How many disk are used for a triangle whose height is not 0?

# Natural Numbers

## Computing Tetrahedral Numbers

- We now proceed to `nth-tri`.
- For every natural number there is a different disk of triangles of different height

(a) Height 1.



(b) Height 2.



(c) Height 3.



- Given that the height is a `natnum`, reason about the structure of a `natnum`
- How many disks are used for a triangle of height 0?
- Height zero means that 0 disks are used
- How many disk are used for a triangle whose height is not 0?
- The disks at each level of the triangle must be added

`tri(1) = 1`

`tri(2) = 1 + 2`

`tri(3) = 1 + 2 + 3`

# Natural Numbers

## Computing Tetrahedral Numbers

- We now proceed to `nth-tri`.
- For every natural number there is a different disk of triangles of different height

(a) Height 1.



(b) Height 2.



(c) Height 3.



- Given that the height is a `natnum`, reason about the structure of a `natnum`
- How many disks are used for a triangle of height 0?
- Height zero means that 0 disks are used
- How many disk are used for a triangle whose height is not 0?
- The disks at each level of the triangle must be added

`tri(1) = 1`

`tri(2) = 1 + 2`

`tri(3) = 1 + 2 + 3`

- `tri(h) = h + tri(h-1)`

# Natural Numbers

## Computing Tetrahedral Numbers

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- ;; Sample expressions for nth-tri  
(define ZERO-VAL-TRI 0)  
(define NINE-VAL-TRI (+ NINE (nth-tri (sub1 NINE))))  
(define SIXTY-VAL-TRI (+ SIXTY (nth-tri (sub1 SIXTY))))

# Natural Numbers

## Computing Tetrahedral Numbers

- `;; natnum → natnum`  
`;; Purpose: Compute the triangular number for the given natnum`  
`(define (nth-tri a-natnum)`
- `;; Sample expressions for nth-tri`  
`(define ZERO-VAL-TRI 0)`  
`(define NINE-VAL-TRI (+ NINE (nth-tri (sub1 NINE))))`  
`(define SIXTY-VAL-TRI (+ SIXTY (nth-tri (sub1 SIXTY))))`

# Natural Numbers

## Computing Tetrahedral Numbers

- ```
;; natnum → natnum
;; Purpose: Compute the triangular number for the given natnum
(define (nth-tri a-natnum)
```
- ```
;; Sample expressions for nth-tri
(define ZERO-VAL-TRI 0)
(define NINE-VAL-TRI (+ NINE (nth-tri (sub1 NINE))))
(define SIXTY-VAL-TRI (+ SIXTY (nth-tri (sub1 SIXTY))))
```
- ```
;; Tests using sample computations for nth-tri
(check-expect (nth-tri ZERO) ZERO-VAL-TRI)
(check-expect (nth-tri NINE) NINE-VAL-TRI)
(check-expect (nth-tri SIXTY) SIXTY-VAL-TRI)
```
- ```
;; Tests using sample values for f-on-natnum
(check-expect (nth-tri 4) 10)
(check-expect (nth-tri 8) 36)
```

# Natural Numbers

## Computing Tetrahedral Numbers

- `;; natnum → natnum`  
`;; Purpose: Compute the triangular number for the given natnum`  
`(define (nth-tri a-natnum)`
- `(if (= a-natnum 0)`  
    `0`  
    `(+ a-natnum (nth-tri (sub1 a-natnum))))`
- `;; Sample expressions for nth-tri`  
`(define ZERO-VAL-TRI 0)`  
`(define NINE-VAL-TRI (+ NINE (nth-tri (sub1 NINE))))`  
`(define SIXTY-VAL-TRI (+ SIXTY (nth-tri (sub1 SIXTY))))`
- `;; Tests using sample computations for nth-tri`  
`(check-expect (nth-tri ZERO) ZERO-VAL-TRI)`  
`(check-expect (nth-tri NINE) NINE-VAL-TRI)`  
`(check-expect (nth-tri SIXTY) SIXTY-VAL-TRI)`

`; Tests using sample values for f-on-natnum`  
`(check-expect (nth-tri 4) 10)`  
`(check-expect (nth-tri 8) 36)`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Natural Numbers

## Homework

- Problems: 169–172

# Interval Processing

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Chapter 5 introduced interval types to describe a value and used them to design functions that make decisions

# Interval Processing

- Chapter 5 introduced interval types to describe a value and used them to design functions that make decisions
- Intervals may be considered compound data that may be processed
- If you think about it carefully processing a natnum,  $n$ , means that all the natural numbers in  $[0..n]$  must be processed

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Interval Processing

- Chapter 5 introduced interval types to describe a value and used them to design functions that make decisions
- Intervals may be considered compound data that may be processed
- If you think about it carefully processing a natnum,  $n$ , means that all the natural numbers in  $[0..n]$  must be processed
- Processing natural numbers fixes the lower end of the interval to 0 while the higher end of the interval is of arbitrary size

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Interval Processing

- Chapter 5 introduced interval types to describe a value and used them to design functions that make decisions
- Intervals may be considered compound data that may be processed
- If you think about it carefully processing a natnum,  $n$ , means that all the natural numbers in  $[0..n]$  must be processed
- Processing natural numbers fixes the lower end of the interval to 0 while the higher end of the interval is of arbitrary size
- We generalize the concept of an interval such that both ends may vary

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Interval Processing

- Chapter 5 introduced interval types to describe a value and used them to design functions that make decisions
- Intervals may be considered compound data that may be processed
- If you think about it carefully processing a natnum,  $n$ , means that all the natural numbers in  $[0..n]$  must be processed
- Processing natural numbers fixes the lower end of the interval to 0 while the higher end of the interval is of arbitrary size
- We generalize the concept of an interval such that both ends may vary
- How do you process an interval of integers?

# Interval Processing

- Chapter 5 introduced interval types to describe a value and used them to design functions that make decisions
- Intervals may be considered compound data that may be processed
- If you think about it carefully processing a natnum,  $n$ , means that all the natural numbers in  $[0..n]$  must be processed
- Processing natural numbers fixes the lower end of the interval to 0 while the higher end of the interval is of arbitrary size
- We generalize the concept of an interval such that both ends may vary
- How do you process an interval of integers?
- Clearly, an interval is data of arbitrary size
- We need a recursive data definition

# Interval Processing

## Interval Data Definition

- Consider what the interval  $[-2..4]$  represents:  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- It represents 7 consecutive integers
- Tells us nothing about the structure of an interval
- What is the smallest interval possible?
- Can an interval contain 0 numbers?

# Interval Processing

## Interval Data Definition

- Consider what the interval  $[-2..4]$  represents:  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- It represents 7 consecutive integers
- Tells us nothing about the structure of an interval
- What is the smallest interval possible?
- Can an interval contain 0 numbers?
- Yes, an interval may be empty
- For example,  $[0..-1]$  is an empty interval

# Interval Processing

## Interval Data Definition

- Consider what the interval  $[-2..4]$  represents:  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- It represents 7 consecutive integers
- Tells us nothing about the structure of an interval
- What is the smallest interval possible?
- Can an interval contain 0 numbers?
- Yes, an interval may be empty
- For example,  $[0..-1]$  is an empty interval
- Given an interval  $[\text{low}..\text{high}]$ , how do we know that the interval is empty?

# Interval Processing

## Interval Data Definition

- Consider what the interval  $[-2..4]$  represents:  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- It represents 7 consecutive integers
- Tells us nothing about the structure of an interval
- What is the smallest interval possible?
- Can an interval contain 0 numbers?
- Yes, an interval may be empty
- For example,  $[0..-1]$  is an empty interval
- Given an interval  $[low..high]$ , how do we know that the interval is empty?
- If  $low$  is greater than  $high$  then the interval is empty

# Interval Processing

## Interval Data Definition

- How do we define a non-empty interval?

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Interval Processing

## Interval Data Definition

- How do we define a non-empty interval?
- Consider  $[-2..4]$ :  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- Here  $\text{low} = -2$  and  $\text{high} = 4$
- Is there any natural way to decompose this interval into two or more parts?

# Interval Processing

## Interval Data Definition

- How do we define a non-empty interval?
- Consider  $[-2..4]$ :  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- Here  $\text{low} = -2$  and  $\text{high} = 4$
- Is there any natural way to decompose this interval into two or more parts?
- Given that the interval is not empty it must have at least one number.
- Let's take that number to be, 4, the high-end of the interval
- We have left:

$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3]$$

# Interval Processing

## Interval Data Definition

- How do we define a non-empty interval?
- Consider  $[-2..4]$ :  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- Here  $\text{low} = -2$  and  $\text{high} = 4$
- Is there any natural way to decompose this interval into two or more parts?
- Given that the interval is not empty it must have at least one number.
- Let's take that number to be, 4, the high-end of the interval
- We have left:  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3]$$
- This is an interval:  $[-2..3]$
- We can rewrite  $[-2..4]$  as follows:  
$$[[-2..3] \ 4]$$
- A non-empty interval has structure: the high number and the subinterval with the rest of the numbers
- Observe that for the subinterval the high end is decremented by 1

# Interval Processing

## Interval Data Definition

- How do we define a non-empty interval?
- Consider  $[-2..4]$ :  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4]$$
- Here  $\text{low} = -2$  and  $\text{high} = 4$
- Is there any natural way to decompose this interval into two or more parts?
- Given that the interval is not empty it must have at least one number.
- Let's take that number to be, 4, the high-end of the interval
- We have left:  
$$[-2 \ -1 \ 0 \ 1 \ 2 \ 3]$$
- This is an interval:  $[-2..3]$
- We can rewrite  $[-2..4]$  as follows:  
$$[[ -2..3 ] \ 4]$$
- A non-empty interval has structure: the high number and the subinterval with the rest of the numbers
- Observe that for the subinterval the high end is decremented by 1
- We can write a data definition for an interval as follows:

An interval ( $[\text{low}..\text{high}]$ ) is two integers such that it is either the:

  1. empty interval (interpretation:  $\text{low} > \text{high}$ )
  2. non-empty interval (interpretation:  $\text{low} \leq \text{high}$ )  
$$[[\text{low}..\text{(sub1 high)}] \ \text{high}]$$
- Suggests that an interval is processed from high down to low until the lower subinterval is empty

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Interval Processing

## Interval Data Definition

- The choice of dividing an interval into high and [low..(sub1 high)] seems rather arbitrary
- Just as easily we can define a non-empty interval as:  
 $[low [(add1 low)..high]]$
- Suggests that an interval is processed from low to high

# Interval Processing

## Interval Data Definition

- The choice of dividing an interval into high and [low..(sub1 high)] seems rather arbitrary
- Just as easily we can define a non-empty interval as:

[low [(add1 low)..high]]

- Suggests that an interval is processed from low to high
- Refine the interval data definition to be:

An interval ([low..high]) is two integers such that it is either:

1. empty interval (interpretation:  $low > high$ )
2. non-empty interval (interpretation:  $low \leq high$ )  
[[low..(sub1 high)] high] or [low [(add1 low)..high]]

# Interval Processing

## Interval Data Definition

- The choice of dividing an interval into high and [low..(sub1 high)] seems rather arbitrary
- Just as easily we can define a non-empty interval as:  
 $[low [(add1 low)..high]]$
- Suggests that an interval is processed from low to high
- Refine the interval data definition to be:  

An interval ([low..high]) is two integers such that it is either:

  1. empty interval (interpretation:  $low > high$ )
  2. non-empty interval (interpretation:  $low \leq high$ )  
[[low..(sub1 high)] high] or [low [(add1 low)..high]]
- This data definition is different from any other we have seen before
- We can choose how to decompose a non-empty interval
- An interval may be processed from high down to low or from low to high

# Interval Processing

## Interval Data Definition

- | # ;; Sample instances of interval  
(define LOW1 ...) (define HIGH1 ...)  
(define LOW2 ...) (define HIGH2 ...) ...  
  
;; [int..int] ... → ...  
;; Purpose: ...  
(define (f-on-interval low high ...)  
 (if (> low high)  
 ...  
 (...high...(f-on-interval low (sub1 high)...  
 ...low....(f-on-interval (add1 low) high...))))  
  
;; Sample expressions for f-on-interval  
(define LOW1-HIGH1-VAL ...)  
(define LOW2-HIGH2-VAL ...)  
...  
;; Tests using sample computations for f-on-interval  
(check-expect (f-on-interval LOW1 HIGH1 ...) LOW1-HIGH1-VAL)  
(check-expect (f-on-interval LOW2 HIGH2 ...) LOW2-HIGH2-VAL) ...  
  
;; Tests using sample values for f-on-interval  
(check-expect (f-on-interval ...) ...)  
(check-expect (f-on-interval ...) ...)

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

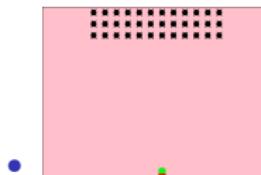
Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Interval Processing

## Creating an Army of Aliens

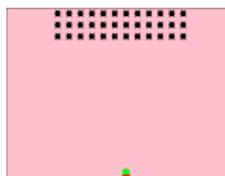
- For the next Aliens Attack version an initial army of aliens must be created



# Interval Processing

## Creating an Army of Aliens

- For the next Aliens Attack version an initial army of aliens must be created



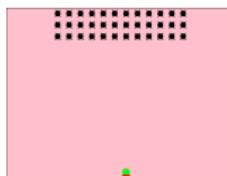
- A list of aliens contains aliens in three different lines
- Each line has the same number of aliens lined up to form columns
- For the army of aliens above:

```
(define ALIEN-LINES 3) (define ALIENS-PER-LINE 12)
```

# Interval Processing

## Creating an Army of Aliens

- For the next Aliens Attack version an initial army of aliens must be created

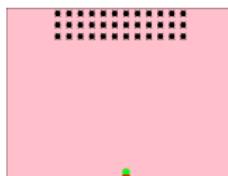


- A list of aliens contains aliens in three different lines
- Each line has the same number of aliens lined up to form columns
- For the army of aliens above:  
`(define ALIEN-LINES 3) (define ALIENS-PER-LINE 12)`
- The aliens start at the top row: `image-y = 0`
- The aliens span the same range of `image-x` values in every line

# Interval Processing

## Creating an Army of Aliens

- For the next Aliens Attack version an initial army of aliens must be created

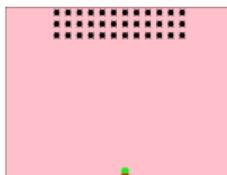


- A list of aliens contains aliens in three different lines
- Each line has the same number of aliens lined up to form columns
- For the army of aliens above:  
`(define ALIEN-LINES 3) (define ALIENS-PER-LINE 12)`
- The aliens start at the top row: `image-y = 0`
- The aliens span the same range of `image-x` values in every line
- The number of alien lines, `num-lines`, is a natural number
- We can process this natural number to create the needed list of aliens
- At each step, a new alien line is added to the list of aliens using the number of lines that still need to be computed to determine the `image-y` value for the aliens in the new line
- Each line of aliens arbitrary size: list of aliens

# Interval Processing

## Creating an Army of Aliens

- For the next Aliens Attack version an initial army of aliens must be created

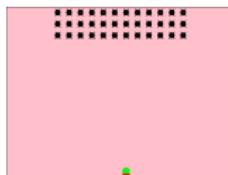


- A list of aliens contains aliens in three different lines
- Each line has the same number of aliens lined up to form columns
- For the army of aliens above:  
`(define ALIEN-LINES 3) (define ALIENS-PER-LINE 12)`
- The aliens start at the top row: `image-y = 0`
- The aliens span the same range of `image-x` values in every line
- The number of alien lines, `num-lines`, is a natural number
- We can process this natural number to create the needed list of aliens
- At each step, a new alien line is added to the list of aliens using the number of lines that still need to be computed to determine the `image-y` value for the aliens in the new line
- Each line of aliens arbitrary size: list of aliens
- Think about the structure of a natural number
- If `num-lines` is 0 then the empty list of aliens is returned
- Otherwise, a line of aliens is created for the range of `image-x` values and added to the list of aliens obtained by recursively processing `num-lines-1`

# Interval Processing

## Creating an Army of Aliens

- For the next Aliens Attack version an initial army of aliens must be created



- A list of aliens contains aliens in three different lines
- Each line has the same number of aliens lined up to form columns
- For the army of aliens above:  
`(define ALIEN-LINES 3) (define ALIENS-PER-LINE 12)`
- The aliens start at the top row: `image-y = 0`
- The aliens span the same range of `image-x` values in every line
- The number of alien lines, `num-lines`, is a natural number
- We can process this natural number to create the needed list of aliens
- At each step, a new alien line is added to the list of aliens using the number of lines that still need to be computed to determine the `image-y` value for the aliens in the new line
- Each line of aliens arbitrary size: list of aliens
- Think about the structure of a natural number
- If `num-lines` is 0 then the empty list of aliens is returned
- Otherwise, a line of aliens is created for the range of `image-x` values and added to the list of aliens obtained by recursively processing `num-lines-1`
- The range of `image-x` values for each line: an interval

# Interval Processing

## Creating an Army of Aliens

- ```
;; Sample instances of natnum (for number of alien lines)
(define ZERO 0)
(define THREE 3)
(define FIVE 5)
```

Interval Processing

Creating an Army of Aliens

- ```
;; Sample instances of natnum (for number of alien lines)
(define ZERO 0)
(define THREE 3)
(define FIVE 5)
```
- Outline of sample expressions

```
;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL (... (make-alien-line image-y low high)
 (create-alien-army (sub1 THREE))))
(define FIVE-VAL (... (make-alien-line image-y low high)
 (create-alien-army (sub1 FIVE))))
```
- What should the *image-y* value be?

# Interval Processing

## Creating an Army of Aliens

- ```
;; Sample instances of natnum (for number of alien lines)
(define ZERO 0)
(define THREE 3)
(define FIVE 5)
```
- Outline of sample expressions

```
;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL (... (make-alien-line image-y low high)
                        (create-alien-army (sub1 THREE))))
(define FIVE-VAL (... (make-alien-line image-y low high)
                      (create-alien-army (sub1 FIVE))))
```
- What should the *image-y* value be?

num-lines	<i>image-y</i> Value
5	4
4	3
3	2
2	1
1	0

Interval Processing

Creating an Army of Aliens

- ```
;; Sample instances of natnum (for number of alien lines)
(define ZERO 0)
(define THREE 3)
(define FIVE 5)
```
- Outline of sample expressions

```
;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL (... (make-alien-line image-y low high)
 (create-alien-army (sub1 THREE))))
(define FIVE-VAL (... (make-alien-line image-y low high)
 (create-alien-army (sub1 FIVE))))
```

- What should the *image-y* value be?

| num-lines | <i>image-y</i> Value |
|-----------|----------------------|
| 5         | 4                    |
| 4         | 3                    |
| 3         | 2                    |
| 2         | 1                    |
| 1         | 0                    |

- What is the pattern?

# Interval Processing

## Creating an Army of Aliens

- ```
;; Sample instances of natnum (for number of alien lines)
(define ZERO 0)
(define THREE 3)
(define FIVE 5)
```
- Outline of sample expressions

```
;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL (... (make-alien-line image-y low high)
                        (create-alien-army (sub1 THREE))))
(define FIVE-VAL (... (make-alien-line image-y low high)
                      (create-alien-army (sub1 FIVE))))
```
- What should the *image-y* value be?

num-lines	<i>image-y</i> Value
5	4
4	3
3	2
2	1
1	0

- What is the pattern?
- The *image-y* value is always one less than *num-lines*
- This means that for each call to *make-alien-line* the needed *image-y* value is (*sub1 num-lines*)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Creating an Army of Aliens

- What is the needed interval for `image-x` values?

Interval Processing

Creating an Army of Aliens

- What is the needed interval for `image-x` values?
- The aliens ought to be relatively centered around:
`(/ MAX-CHARS-HORIZONTAL 2)`
- Half the aliens ought to be placed before the midpoint and half after the midpoint
- Interval definition for `image-x` values:
`(define ALIEN-LINE-XLOW (- (/ MAX-CHARS-HORIZONTAL 2)
 (/ ALIENS-PER-LINE 2)))
(define ALIEN-LINE-XHIGH (sub1 (+ (/ MAX-CHARS-HORIZONTAL 2)
 (/ ALIENS-PER-LINE 2))))`
- Half the number of aliens per line is added and subtracted from the midpoint making the interval one number too big
- Therefore, 1 is subtracted from the high end of the interval (instead, 1 could be subtracted from the low end of the interval)

Interval Processing

Creating an Army of Aliens

- Use cons combine an alien line with the rest of the aliens?

```
(cons alien line
  (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0)))
  rest of the aliens
  (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
    (make-posn 9 2) (make-posn 10 2)))
```

Interval Processing

Creating an Army of Aliens

- Use cons combine an alien line with the rest of the aliens?

```
(cons alien line
  (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
    rest of the aliens
    (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
      (make-posn 9 2) (make-posn 10 2)))
```

- The resulting list is:

```
(list (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
  (make-posn 9 1)
  (make-posn 10 1)
  (make-posn 11 1)      This is not a list of aliens
  (make-posn 9 2)
  (make-posn 10 2))
```

- We need a function that combines two lists into one list

Interval Processing

Creating an Army of Aliens

- Use cons combine an alien line with the rest of the aliens?

```
(cons alien line
  (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
    rest of the aliens
    (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
      (make-posn 9 2) (make-posn 10 2)))
```

- The resulting list is:

```
(list (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
  (make-posn 9 1)
  (make-posn 10 1)
  (make-posn 11 1)      This is not a list of aliens
  (make-posn 9 2)
  (make-posn 10 2))
```

- We need a function that combines two lists into one list
- **append**: $(\text{listof } X)^* \rightarrow (\text{listof } X)$

Interval Processing

Creating an Army of Aliens

- Use cons combine an alien line with the rest of the aliens?

```
(cons alien line
  (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
    rest of the aliens
    (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
      (make-posn 9 2) (make-posn 10 2)))
```

- The resulting list is:

```
(list (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
  (make-posn 9 1)
  (make-posn 10 1)
  (make-posn 11 1)      This is not a list of aliens
  (make-posn 9 2)
  (make-posn 10 2))
```

- We need a function that combines two lists into one list

- `append: (listof X)* → (listof X)`

- Consider:

```
(append (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
  (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
    (make-posn 9 2) (make-posn 10 2)))
```

Interval Processing

Creating an Army of Aliens

- Use cons combine an alien line with the rest of the aliens?

```
(cons alien line
  (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
    rest of the aliens
    (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
      (make-posn 9 2) (make-posn 10 2)))
```

- The resulting list is:

```
(list (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
  (make-posn 9 1)
  (make-posn 10 1)
  (make-posn 11 1)      This is not a list of aliens
  (make-posn 9 2)
  (make-posn 10 2))
```

- We need a function that combines two lists into one list

- `append: (listof X)* → (listof X)`

- Consider:

```
(append (list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0))
  (list (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
    (make-posn 9 2) (make-posn 10 2)))
```

- The resulting list is a list of aliens:

```
(list (make-posn 9 0) (make-posn 10 0) (make-posn 11 0)
  (make-posn 9 1) (make-posn 10 1) (make-posn 11 1)
  (make-posn 9 2) (make-posn 10 2))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Creating an Army of Aliens

- ```
;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL
 (append (make-alien-line (sub1 THREE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 THREE))))
(define FIVE-VAL
 (append (make-alien-line (sub1 FIVE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 FIVE))))
```

# Interval Processing

## Creating an Army of Aliens

- ```
;; natnum → loa  Purpose: Create initial alien army
(define (create-alien-army num-lines)
```
- ```
;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL
 (append (make-alien-line (sub1 THREE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 THREE))))
(define FIVE-VAL
 (append (make-alien-line (sub1 FIVE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 FIVE))))
```

# Interval Processing

## Creating an Army of Aliens

- ```
;; natnum → loa  Purpose: Create initial alien army
(define (create-alien-army num-lines)
```
- ```
; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL
 (append (make-alien-line (sub1 THREE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 THREE))))
(define FIVE-VAL
 (append (make-alien-line (sub1 FIVE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 FIVE))))
```
- ```
; Tests using sample computations for create-alien-army
(check-expect (create-alien-army ZERO) ZERO-VAL)
(check-expect (create-alien-army THREE) THREE-VAL)
(check-expect (create-alien-army FIVE) FIVE-VAL)
; Tests using sample values for create-alien-army
(check-expect (create-alien-army 1)
              (list (make-posn 4 0) (make-posn 5 0) (make-posn 6 0)
                    (make-posn 7 0) (make-posn 8 0) (make-posn 9 0)
                    (make-posn 10 0) (make-posn 11 0) (make-posn 12 0)
                    (make-posn 13 0) (make-posn 14 0) (make-posn 15 0)))
```

Interval Processing

Creating an Army of Aliens

- ```
;; natnum → loa Purpose: Create initial alien army
(define (create-alien-army num-lines)
 (if (= num-lines 0)
 '()
 (append
 (make-alien-line (sub1 num-lines) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 num-lines)))))

;; Sample expressions for create-alien-army
(define ZERO-VAL '())
(define THREE-VAL
 (append (make-alien-line (sub1 THREE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 THREE))))
(define FIVE-VAL
 (append (make-alien-line (sub1 FIVE) ALIEN-LINE-XLOW ALIEN-LINE-XHIGH)
 (create-alien-army (sub1 FIVE))))
```
- ```
; Tests using sample computations for create-alien-army
(check-expect (create-alien-army ZERO) ZERO-VAL)
(check-expect (create-alien-army THREE) THREE-VAL)
(check-expect (create-alien-army FIVE) FIVE-VAL)
; Tests using sample values for create-alien-army
(check-expect (create-alien-army 1)
              (list (make-posn 4 0) (make-posn 5 0) (make-posn 6 0)
                    (make-posn 7 0) (make-posn 8 0) (make-posn 9 0)
                    (make-posn 10 0) (make-posn 11 0) (make-posn 12 0)
                    (make-posn 13 0) (make-posn 14 0) (make-posn 15 0)))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Creating an Army of Aliens

- `make-alien-line` must traverse the given interval
- Think in terms of the structure of an interval

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Creating an Army of Aliens

- `make-alien-line` must traverse the given interval
- Think in terms of the structure of an interval
- If the interval is empty the empty list of aliens is returned

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Creating an Army of Aliens

- `make-alien-line` must traverse the given interval
- Think in terms of the structure of an interval
- If the interval is empty the empty list of aliens is returned
- Otherwise, a new alien is created using low end value of the given interval and the given `image-y` value
- This new alien is added to the front of the list of aliens obtained from processing the rest of the interval

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Creating an Army of Aliens

- ;; Sample instances of interval

```
(define LOW1 THREE) (define HIGH1 ZERO)
(define LOW2 ZERO) (define HIGH2 THREE)
(define LOW3 THREE) (define HIGH3 FIVE)
```


;; Sample expressions for make-alien-line

```
(define LOW1-HIGH1-VAL '())
(define LOW2-HIGH2-VAL (cons (make-posn LOW2 3)
                               (make-alien-line 3 (add1 LOW2) HIGH2)))
(define LOW3-HIGH3-VAL (cons (make-posn LOW3 2)
                               (make-alien-line 2 (add1 LOW3) HIGH3)))
```

Interval Processing

Creating an Army of Aliens

- ```
;; image-y [image-x..image-x] → loa
;; Purpose: Create an loa with an alien for each value in
;; the given interval using the given image-y
(define (make-alien-line an-image-y low high))
```
- ```
;; Sample instances of interval
(define LOW1 THREE) (define HIGH1 ZERO)
(define LOW2 ZERO) (define HIGH2 THREE)
(define LOW3 THREE) (define HIGH3 FIVE)

;; Sample expressions for make-alien-line
(define LOW1-HIGH1-VAL '())
(define LOW2-HIGH2-VAL (cons (make-posn LOW2 3)
                             (make-alien-line 3 (add1 LOW2) HIGH2)))
(define LOW3-HIGH3-VAL (cons (make-posn LOW3 2)
                             (make-alien-line 2 (add1 LOW3) HIGH3)))
```

Interval Processing

Creating an Army of Aliens

- ```
;; image-y [image-x..image-x] → loa
;; Purpose: Create an loa with an alien for each value in
;; the given interval using the given image-y
(define (make-alien-line an-image-y low high))
```
- ```
;; Sample instances of interval
(define LOW1 THREE) (define HIGH1 ZERO)
(define LOW2 ZERO) (define HIGH2 THREE)
(define LOW3 THREE) (define HIGH3 FIVE)

;; Sample expressions for make-alien-line
(define LOW1-HIGH1-VAL '())
(define LOW2-HIGH2-VAL (cons (make-posn LOW2 3)
                             (make-alien-line 3 (add1 LOW2) HIGH2)))
(define LOW3-HIGH3-VAL (cons (make-posn LOW3 2)
                             (make-alien-line 2 (add1 LOW3) HIGH3)))
```
- ```
;; Tests using sample computations for make-alien-line
(check-expect (make-alien-line 3 LOW2 HIGH2) LOW2-HIGH2-VAL)
(check-expect (make-alien-line 2 LOW3 HIGH3) LOW3-HIGH3-VAL)

;; Tests using sample values for make-alien-line
(check-expect (make-alien-line 1 17 19)
 (list (make-posn 17 1)
 (make-posn 18 1)
 (make-posn 19 1))))
```

# Interval Processing

## Creating an Army of Aliens

- ```
;; image-y [image-x..image-x] → loa
;; Purpose: Create an loa with an alien for each value in
;;           the given interval using the given image-y
(define (make-alien-line an-image-y low high)
  (if (> low high)
      '()
      (cons (make-posn low an-image-y)
            (make-alien-line an-image-y (add1 low) high))))
```
- ```
;; Sample instances of interval
(define LOW1 THREE) (define HIGH1 ZERO)
(define LOW2 ZERO) (define HIGH2 THREE)
(define LOW3 THREE) (define HIGH3 FIVE)

;; Sample expressions for make-alien-line
(define LOW1-HIGH1-VAL '())
(define LOW2-HIGH2-VAL (cons (make-posn LOW2 3)
 (make-alien-line 3 (add1 LOW2) HIGH2)))
(define LOW3-HIGH3-VAL (cons (make-posn LOW3 2)
 (make-alien-line 2 (add1 LOW3) HIGH3)))
```
- ```
;; Tests using sample computations for make-alien-line
(check-expect (make-alien-line 3 LOW2 HIGH2) LOW2-HIGH2-VAL)
(check-expect (make-alien-line 2 LOW3 HIGH3) LOW3-HIGH3-VAL)

;; Tests using sample values for make-alien-line
(check-expect (make-alien-line 1 17 19)
              (list (make-posn 17 1)
                    (make-posn 18 1)
                    (make-posn 19 1))))
```

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?
- Consider finding the largest prime in [9..22]

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?
- Consider finding the largest prime in [9..22]
- If processed from low to high you discover that 9 and 10 are not prime
- Think of the interval as divided into two pieces:

[9..10]: interval has no primes

[11..22]: interval not explored and may have a largest prime

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?
- Consider finding the largest prime in [9..22]
- If processed from low to high you discover that 9 and 10 are not prime
- Think of the interval as divided into two pieces:

[9..10]: interval has no primes

[11..22]: interval not explored and may have a largest prime

- In the next step you determine that 11 is prime
- You may think of the interval in three pieces:

[9..10]: interval has no primes

[11..11]: 11 is the smallest prime

[12..22]: interval not explored and may have a larger prime

- How do you determine if 11 is the largest prime?

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?
- Consider finding the largest prime in [9..22]
- If processed from low to high you discover that 9 and 10 are not prime
- Think of the interval as divided into two pieces:

[9..10]: interval has no primes

[11..22]: interval not explored and may have a largest prime

- In the next step you determine that 11 is prime
- You may think of the interval in three pieces:

[9..10]: interval has no primes

[11..11]: 11 is the smallest prime

[12..22]: interval not explored and may have a larger prime

- How do you determine if 11 is the largest prime?
- You need to find, if it exists, the largest prime in [12..22]

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?
- Consider finding the largest prime in [9..22]
- If processed from low to high you discover that 9 and 10 are not prime
- Think of the interval as divided into two pieces:

[9..10]: interval has no primes

[11..22]: interval not explored and may have a largest prime

- In the next step you determine that 11 is prime
- You may think of the interval in three pieces:

[9..10]: interval has no primes

[11..11]: 11 is the smallest prime

[12..22]: interval not explored and may have a larger prime

- How do you determine if 11 is the largest prime?
- You need to find, if it exists, the largest prime in [12..22]
- Consider processing the interval from high down to low:

[20..22]: interval has no primes

[9..19]: interval not explored and may have a largest prime

Interval Processing

Largest Prime in an Interval

- Consider the problem of finding the largest prime number in a given interval
- Process low to high or high down to low?
- Does it matter which one you choose?
- Consider finding the largest prime in [9..22]
- If processed from low to high you discover that 9 and 10 are not prime
- Think of the interval as divided into two pieces:
 - [9..10]: interval has no primes
 - [11..22]: interval not explored and may have a largest prime
- In the next step you determine that 11 is prime
- You may think of the interval in three pieces:
 - [9..10]: interval has no primes
 - [11..11]: 11 is the smallest prime
 - [12..22]: interval not explored and may have a larger prime
- How do you determine if 11 is the largest prime?
- You need to find, if it exists, the largest prime in [12..22]
- Consider processing the interval from high down to low:
 - [20..22]: interval has no primes
 - [9..19]: interval not explored and may have a largest prime
- In the next step you determine that 19 is a prime:
 - [20..22]: interval has no primes
 - [19..19]: 19 is the largest prime
 - [9..18]: interval not explored and may have a largest prime
- You immediately know that the first prime found is the largest one

Interval Processing

Largest Prime in an Interval

- ;; Sample instances of interval
 - (define LOW1 2) (define HIGH1 1)
 - (define LOW2 2) (define HIGH2 23)
 - (define LOW3 13) (define HIGH3 16)

Interval Processing

Largest Prime in an Interval

- ;; Sample instances of interval
(define LOW1 2) (define HIGH1 1)
(define LOW2 2) (define HIGH2 23)
(define LOW3 13) (define HIGH3 16)

- ;; Sample expressions for largest-prime
(define LOW1-HIGH1-VAL -1)
(define LOW2-HIGH2-VAL HIGH2)
(define LOW3-HIGH3-VAL (largest-prime LOW3 (sub1 HIGH3)))

Interval Processing

Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW1 2) (define HIGH1 1)
(define LOW2 2) (define HIGH2 23)
(define LOW3 13) (define HIGH3 16)
```
- ```
;; [int..int] → int
;; Purpose: Return largest prime in interval. If none, return -1
;; ASSUMPTION: low > 1
(define (largest-prime low high))
```
- ```
;; Sample expressions for largest-prime
(define LOW1-HIGH1-VAL -1)
(define LOW2-HIGH2-VAL HIGH2)
(define LOW3-HIGH3-VAL (largest-prime LOW3 (sub1 HIGH3)))
```

# Interval Processing

## Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW1 2)  (define HIGH1 1)
(define LOW2 2)  (define HIGH2 23)
(define LOW3 13) (define HIGH3 16)
```
- ```
;; [int..int] → int
;; Purpose: Return largest prime in interval. If none, return -1
;; ASSUMPTION: low > 1
(define (largest-prime low high))
```

- ```
;; Sample expressions for largest-prime
(define LOW1-HIGH1-VAL -1)
(define LOW2-HIGH2-VAL HIGH2)
(define LOW3-HIGH3-VAL (largest-prime LOW3 (sub1 HIGH3)))
```
- ```
;; Tests using sample computations for largest-prime
(check-expect (largest-prime LOW1 HIGH1) LOW1-HIGH1-VAL)
(check-expect (largest-prime LOW2 HIGH2) LOW2-HIGH2-VAL)
(check-expect (largest-prime LOW3 HIGH3) LOW3-HIGH3-VAL)
;; Tests using sample values for largest-prime
(check-expect (largest-prime 2 2) 2)
(check-expect (largest-prime 24 28) -1)
(check-expect (largest-prime 29 35) 31)
```

# Interval Processing

## Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW1 2)  (define HIGH1 1)
(define LOW2 2)  (define HIGH2 23)
(define LOW3 13) (define HIGH3 16)
```
- ```
;; [int..int] → int
;; Purpose: Return largest prime in interval. If none, return -1
;; ASSUMPTION: low > 1
(define (largest-prime low high))
```
- ```
(cond [(> low high) -1]
        [(not (is-divisible? high 2 (quotient high 2))) high]
        [else (largest-prime low (sub1 high))]))
```
- ```
;; Sample expressions for largest-prime
(define LOW1-HIGH1-VAL -1)
(define LOW2-HIGH2-VAL HIGH2)
(define LOW3-HIGH3-VAL (largest-prime LOW3 (sub1 HIGH3)))
```
- ```
;; Tests using sample computations for largest-prime
(check-expect (largest-prime LOW1 HIGH1) LOW1-HIGH1-VAL)
(check-expect (largest-prime LOW2 HIGH2) LOW2-HIGH2-VAL)
(check-expect (largest-prime LOW3 HIGH3) LOW3-HIGH3-VAL)
;; Tests using sample values for largest-prime
(check-expect (largest-prime 2 2) 2)
(check-expect (largest-prime 24 28) -1)
(check-expect (largest-prime 29 35) 31)
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- Lesson: explore different designs
- One design led to a simpler solution to the problem
- A good problem solver explores different designs, whenever possible, before committing to one

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- `is-divisible?` processes an interval

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- `is-divisible?` processes an interval
- Once again, we must decide in what direction to process the given interval

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- `is-divisible?` processes an interval
- Once again, we must decide in what direction to process the given interval
- If the given natural number is not divisible by any number in the given interval then the entire interval must be processed to determine this
- The direction of the processing uses makes no difference
- If the given number is divisible the search for a factor may stop when any factor is found

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- `is-divisible?` processes an interval
- Once again, we must decide in what direction to process the given interval
- If the given natural number is not divisible by any number in the given interval then the entire interval must be processed to determine this
- The direction of the processing uses makes no difference
- If the given number is divisible the search for a factor may stop when any factor is found
- Arbitrarily process from `low` to `high`
- Observe that this is an ORing operation on an interval (not a list)
- The given natural number is divisible if the given interval is not empty and the interval's low-end number divides the given natural number or a number in the rest of the interval divides the given natural number
- A conditional expression, as suggested by the template for a function on an interval, is not needed

Interval Processing

Largest Prime in an Interval

- `;; Sample instances of interval
(define LOW4 21) (define HIGH4 20)
(define LOW5 2) (define HIGH5 13)
(define LOW6 2) (define HIGH6 8)`

Interval Processing

Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW4 21) (define HIGH4 20)
(define LOW5 2) (define HIGH5 13)
(define LOW6 2) (define HIGH6 8)
```
- ```
;; Sample expressions for is-divisible?
(define LOW4-HIGH4-VAL
  (and (not (> LOW4 HIGH4))
       (or (= (remainder 11 LOW4) 0)
           (is-divisible? 11 (add1 LOW4) HIGH4))))
(define LOW5-HIGH5-VAL
  (and (not (> LOW5 HIGH5))
       (or (= (remainder 27 LOW5) 0)
           (is-divisible? 27 (add1 LOW5) HIGH5))))
(define LOW6-HIGH6-VAL
  (and (not (> LOW6 HIGH6))
       (or (= (remainder 17 LOW6) 0)
           (is-divisible? 17 (add1 LOW6) HIGH6))))
```

Interval Processing

Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW4 21) (define HIGH4 20)
(define LOW5 2) (define HIGH5 13)
(define LOW6 2) (define HIGH6 8)
```
- ```
;; natnum [natnum..natnum] → Boolean
;; Purpose: Determine if the given natnum is divisible by
;;           any natnum in the given interval
(define (is-divisible? target low high)
```
- ```
;; Sample expressions for is-divisible?
(define LOW4-HIGH4-VAL
 (and (not (> LOW4 HIGH4))
 (or (= (remainder 11 LOW4) 0)
 (is-divisible? 11 (add1 LOW4) HIGH4))))
(define LOW5-HIGH5-VAL
 (and (not (> LOW5 HIGH5))
 (or (= (remainder 27 LOW5) 0)
 (is-divisible? 27 (add1 LOW5) HIGH5))))
(define LOW6-HIGH6-VAL
 (and (not (> LOW6 HIGH6))
 (or (= (remainder 17 LOW6) 0)
 (is-divisible? 17 (add1 LOW6) HIGH6))))
```

# Interval Processing

## Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW4 21)  (define HIGH4 20)
(define LOW5 2)   (define HIGH5 13)
(define LOW6 2)   (define HIGH6 8)
```
- ```
;; natnum [natnum..natnum] → Boolean
;; Purpose: Determine if the given natnum is divisible by
;; any natnum in the given interval
(define (is-divisible? target low high)
```
- ```
;; Sample expressions for is-divisible?
(define LOW4-HIGH4-VAL
  (and (not (> LOW4 HIGH4))
       (or (= (remainder 11 LOW4) 0)
           (is-divisible? 11 (add1 LOW4) HIGH4))))
(define LOW5-HIGH5-VAL
  (and (not (> LOW5 HIGH5))
       (or (= (remainder 27 LOW5) 0)
           (is-divisible? 27 (add1 LOW5) HIGH5))))
(define LOW6-HIGH6-VAL
  (and (not (> LOW6 HIGH6))
       (or (= (remainder 17 LOW6) 0)
           (is-divisible? 17 (add1 LOW6) HIGH6))))
• ; Tests using sample computations for is-divisible?
(check-expect (is-divisible? 11 LOW4 HIGH4) LOW4-HIGH4-VAL)
(check-expect (is-divisible? 27 LOW5 HIGH5) LOW5-HIGH5-VAL)
(check-expect (is-divisible? 17 LOW6 HIGH6) LOW6-HIGH6-VAL)
;; Tests using sample values for largest-prime
(check-expect (is-divisible? 2 2 2) #true)
(check-expect (is-divisible? 51 2 25) #true)
(check-expect (is-divisible? 53 2 26) #false)
```

Interval Processing

Largest Prime in an Interval

- ```
;; Sample instances of interval
(define LOW4 21) (define HIGH4 20)
(define LOW5 2) (define HIGH5 13)
(define LOW6 2) (define HIGH6 8)
```
- ```
;; natnum [natnum..natnum] → Boolean
;; Purpose: Determine if the given natnum is divisible by
;;           any natnum in the given interval
(define (is-divisible? target low high)
  (and (not (> low high))
       (or (= (remainder target low) 0)
           (is-divisible? target (add1 low) high))))
```
- ```
; Sample expressions for is-divisible?
(define LOW4-HIGH4-VAL
 (and (not (> LOW4 HIGH4))
 (or (= (remainder 11 LOW4) 0)
 (is-divisible? 11 (add1 LOW4) HIGH4))))
(define LOW5-HIGH5-VAL
 (and (not (> LOW5 HIGH5))
 (or (= (remainder 27 LOW5) 0)
 (is-divisible? 27 (add1 LOW5) HIGH5))))
(define LOW6-HIGH6-VAL
 (and (not (> LOW6 HIGH6))
 (or (= (remainder 17 LOW6) 0)
 (is-divisible? 17 (add1 LOW6) HIGH6))))
```
- ```
; Tests using sample computations for is-divisible?
(check-expect (is-divisible? 11 LOW4 HIGH4) LOW4-HIGH4-VAL)
(check-expect (is-divisible? 27 LOW5 HIGH5) LOW5-HIGH5-VAL)
(check-expect (is-divisible? 17 LOW6 HIGH6) LOW6-HIGH6-VAL)
;; Tests using sample values for largest-prime
(check-expect (is-divisible? 2 2 2) #true)
(check-expect (is-divisible? 51 2 25) #true)
(check-expect (is-divisible? 53 2 26) #false)
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Largest Prime in an Interval

- Lesson: A problem may require processing several intervals
- Usually, each interval that needs to be processed requires a different function to be designed
- Different intervals do not have to be processed in the same direction

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Interval Processing

Homework

- Problems: 173–176

Aliens Attack Version 4

- Refinement: multiple aliens and shots
- The first step is to refine the data definition for a world and, consequently, the template for functions on a world
- This will lead us to the refinements needed to update the game.

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

New world Data Definition and Function Template

- The data definitions for a list of aliens (`loa`) and a list of shots (`los`) were defined in Section 67

Aliens Attack Version 4

New world Data Definition and Function Template

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- The data definitions for a list of aliens (`loa`) and a list of shots (`los`) were defined in Section 67
- `;; A world is a structure: (make-world rocket loa dir los)`
`(define-struct world (rocket aliens dir shots))`
- Observe that the names of the selectors change
- We do so to make sure the structure's field names convey to any reader of our code what they represent

Aliens Attack Version 4

New world Data Definition and Function Template

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- The data definitions for a list of aliens (`loa`) and a list of shots (`los`) were defined in Section 67
- `;; A world is a structure: (make-world rocket loa dir los)`
`(define-struct world (rocket aliens dir shots))`
- Observe that the names of the selectors change
- We do so to make sure the structure's field names convey to any reader of our code what they represent
- This refinement also means that the template for a function on a `world` must also be refined
- Specifically, a function on a `world` should no longer call a function on an `alien` or a function on a `shot`
- Instead, it must call functions on a `loa` and on a `los`

Aliens Attack Version 4

New world Data Definition and Function Template

- ```
;; world ... → ... Purpose: ...
(define (f-on-world a-world ...)
 ... (f-on-rocket (world-rocket a-world) ...)
 ... (f-on-loa (world-alien a-world) ...)
 ... (f-on-dir (world-dir a-world) ...)
 ... (f-on-los (world-shots a-world) ...))

;; Sample instances for world
(define WORLD1 (make-world ...))

⋮

;; Sample expressions for f-on-world
(define WORLD-VAL1 ...)

⋮

;; Tests using sample computations for f-on-world
(check-expect (f-on-world WORLD1 ...) WORLD-VAL1)

⋮

;; Tests using sample values for f-on-world
(check-expect (f-on-world ...) ...)

⋮
```

# Aliens Attack Version 4

## New world Data Definition and Function Template

- The sample worlds may now be updated as follows:

```
(define INIT-LOA (create-alien-army ALIEN-LINES))
```

```
(define INIT-WORLD (make-world INIT-ROCKET
INIT-LOA
INIT-DIR
E-LOS))
```

```
(define INIT-WORLD2 (make-world INIT-ROCKET2
(list INIT-ALIEN2)
DIR2
(list SHOT2)))
```

```
(define WORLD3 (make-world 7
(list (make-posn 3 3))
'right
(list (make-posn 3 3))))
```

- The initial list of aliens, INIT-LOA, is defined using the `create-alien-army` developed in Section 87
- The initial list of shots is defined in Section 67
- The remaining two worlds are updated to contain the single alien and the single shot used in Aliens Attack 3

# Aliens Attack Version 4

## New world Data Definition and Function Template

- The sample worlds may now be updated as follows:

```
(define INIT-LOA (create-alien-army ALIEN-LINES))
```

```
(define INIT-WORLD (make-world INIT-ROCKET
INIT-LOA
INIT-DIR
E-LOS))
```

```
(define INIT-WORLD2 (make-world INIT-ROCKET2
(list INIT-ALIEN2)
DIR2
(list SHOT2)))
```

```
(define WORLD3 (make-world 7
(list (make-posn 3 3))
'right
(list (make-posn 3 3))))
```

- The initial list of aliens, INIT-LOA, is defined using the `create-alien-army` developed in Section 87
- The initial list of shots is defined in Section 67
- The remaining two worlds are updated to contain the single alien and the single shot used in Aliens Attack 3
- run function and big-bang expression remain unchanged

# Aliens Attack Version 4

## The draw-world Refinement

- Drawing the world now means drawing a `loa` and a `los`
- The sample expressions for `draw-world` must be refined

# Aliens Attack Version 4

## The draw-world Refinement

- Drawing the world now means drawing a `loa` and a `los`
- The sample expressions for `draw-world` must be refined
- Two new problems must be solved: drawing a list of aliens and drawing a list of shots
- We assume these functions exist and that they need as input the proper list to draw and the scene to draw in

# Aliens Attack Version 4

## The draw-world Refinement

- ;; Sample expressions for draw-world

```
(define WORLD-SCN1
 (draw-los (world-shots INIT-WORLD)
 (draw-loa (world-alien INIT-WORLD)
 (draw-rocket (world-rocket INIT-WORLD) E-SCENE))))
(define WORLD-SCN2
 (draw-los (world-shots INIT-WORLD2)
 (draw-loa (world-alien INIT-WORLD2)
 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))
```

# Aliens Attack Version 4

## The draw-world Refinement

- ;; world → scene Purpose: To draw the world in E-SCENE  
(define (draw-world a-world)

- ;; Sample expressions for draw-world  
(define WORLD-SCN1  
 (draw-los (world-shots INIT-WORLD)  
 (draw-loa (world-alien INIT-WORLD)  
 (draw-rocket (world-rocket INIT-WORLD) E-SCENE))))  
(define WORLD-SCN2  
 (draw-los (world-shots INIT-WORLD2)  
 (draw-loa (world-alien INIT-WORLD2)  
 (draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))

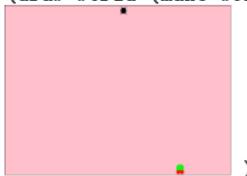
# Aliens Attack Version 4

## The draw-world Refinement

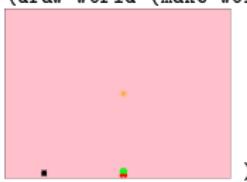
- `;; world → scene Purpose: To draw the world in E-SCENE`  
`(define (draw-world a-world)`

- `;; Sample expressions for draw-world`  
`(define WORLD-SCN1`  
 `(draw-loa (world-shots INIT-WORLD))`  
 `(draw-loa (world-alien INIT-WORLD))`  
 `(draw-rocket (world-rocket INIT-WORLD) E-SCENE))))`  
`(define WORLD-SCN2`  
 `(draw-loa (world-shots INIT-WORLD2))`  
 `(draw-loa (world-alien INIT-WORLD2))`  
 `(draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))`
- `;; Tests using sample computations for draw-world`  
`(check-expect`

```
(draw-world (make-world INIT-ROCKET2 (list INIT-ALIEN) DIR3 '()))
```

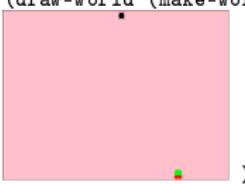
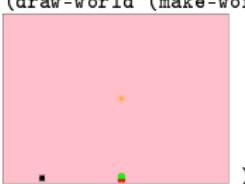


```
(check-expect
(draw-world (make-world INIT-ROCKET (list INIT-ALIEN2) DIR2 (list SHOT2)))
```



# Aliens Attack Version 4

## The draw-world Refinement

- `;; world → scene Purpose: To draw the world in E-SCENE`  
`(define (draw-world a-world)`  
 `(draw-lost (world-shots a-world)`  
 `(draw-loa (world-alien a-world)`  
 `(draw-rocket (world-rocket a-world) E-SCENE))))`
- `;; Sample expressions for draw-world`  
`(define WORLD-SCN1`  
 `(draw-lost (world-shots INIT-WORLD)`  
 `(draw-loa (world-alien INIT-WORLD)`  
 `(draw-rocket (world-rocket INIT-WORLD) E-SCENE))))`
- `(define WORLD-SCN2`  
 `(draw-lost (world-shots INIT-WORLD2)`  
 `(draw-loa (world-alien INIT-WORLD2)`  
 `(draw-rocket (world-rocket INIT-WORLD2) E-SCENE))))`
- `;; Tests using sample computations for draw-world`  
`(check-expect`  
 `(draw-world (make-world INIT-ROCKET2 (list INIT-ALIEN) DIR3 '()))`  
  
`)`  
`(check-expect`  
 `(draw-world (make-world INIT-ROCKET (list INIT-ALIEN2) DIR2 (list SHOT2)))`  


# Aliens Attack Version 4

## The draw-world Refinement

- To draw the aliens a list of aliens must be processed
- Reason about the structure of a loa

# Aliens Attack Version 4

## The draw-world Refinement

- To draw the aliens a list of aliens must be processed
- Reason about the structure of a loa
- If the given list of aliens is empty the result is the given scene

# Aliens Attack Version 4

## The draw-world Refinement

- To draw the aliens a list of aliens must be processed
- Reason about the structure of a loa
- If the given list of aliens is empty the result is the given scene
- If the given list of aliens is not empty then the rest of the aliens are recursively processed to obtain a scene that contains all the aliens except the first
- In this scene the first alien is drawn to obtain a scene that contains all the aliens



## Aliens Attack Version 4

## The draw-world Refinement

- `; loa scene → scene` Purpose: Draw loa in the given scene  
`(define (draw-loa a-loa scn)`

## Aliens Attack Version 4

## The draw-world Refinement

- `;; loa scene → scene` Purpose: Draw loa in the given scene  
`(define (draw-loa a-loa scn)`

## Lists

# List Processing

# Natural Numbers

# Interval Processing

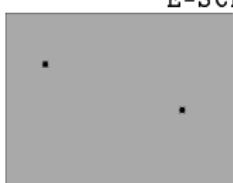
# Aliens Attack

## Version 4

## Binary Trees

## Mutually Recursive Data

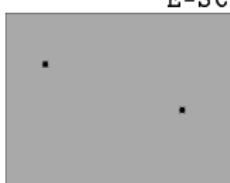
## Processing Multiple Inputs of Arbitrary Size



# Aliens Attack Version 4

## The draw-world Refinement

- `; ; loa scene → scene` Purpose: Draw loa in the given scene  
`(define (draw-loa a-loa scn)`
- `(if (empty? a-loa)`  
    `scn`  
    `(draw-alien (first a-loa)`  
    `(draw-loa (rest a-loa) scn))))`
- `; ; Sample expressions for draw-loa`  
`(define ELOA-VAL E-SCENE)`  
`(define ILOA-VAL (draw-alien (first INIT-LOA)`  
    `(draw-loa (rest INIT-LOA)`  
    `E-SCENE)))`
- `; ; Tests using sample computations for draw-loa`  
`(check-expect (draw-loa E-LOA E-SCENE) ELOA-VAL)`  
`(check-expect (draw-loa INIT-LOA E-SCENE) ILOA-VAL)`  
`; ; Tests using sample values for draw-loa`  
`(check-expect (draw-loa (cons (make-posn 3 4)`  
    `(cons (make-posn 15 8) '()))`  
    `E-SCENE2)`



)

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The draw-world Refinement

- To draw the shots a list of shots must be processed
- Reason about the structure of a los

# Aliens Attack Version 4

## The draw-world Refinement

- To draw the shots a list of shots must be processed
- Reason about the structure of a los
- If the given list of shots the result is the given scene

# Aliens Attack Version 4

## The draw-world Refinement

- To draw the shots a list of shots must be processed
- Reason about the structure of a los
- If the given list of shots the result is the given scene
- If the given los is not empty then the rest of the shots are recursively processed to obtain a scene that contains all the shots except the first
- In this scene the first shot is drawn to obtain a scene that contains all the shots

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The draw-world Refinement

- ;; Sample expressions for draw-los  
(define ELOS-VAL E-SCENE)  
(define ALOS-VAL (draw-shot (first A-LOS)  
                                      (draw-los (rest A-LOS) E-SCENE)))

# Aliens Attack Version 4

## The draw-world Refinement

- ```
;; los scene → scene
;; Purpose: To draw the given los in the given scene
(define (draw-los a-los scn)
```
- ```
;; Sample expressions for draw-los
(define ELOS-VAL E-SCENE)
(define ALOS-VAL (draw-shot (first A-LOS)
 (draw-los (rest A-LOS) E-SCENE)))
```

# Aliens Attack Version 4

## The draw-world Refinement

- ```
;; los scene → scene
;; Purpose: To draw the given los in the given scene
(define (draw-los a-los scn)
```
- ```
;; Sample expressions for draw-los
(define ELOS-VAL E-SCENE)
(define ALOS-VAL (draw-shot (first A-LOS)
 (draw-los (rest A-LOS) E-SCENE)))
;; Tests using sample computations for draw-los
(check-expect (draw-los E-LOS E-SCENE) ELOS-VAL)
(check-expect (draw-los A-LOS E-SCENE) ALOS-VAL)
;; Tests using sample values for draw-los
(check-expect (draw-los (cons (make-posn 14 8)
 (cons (make-posn 3 2) '())))
 E-SCENE2)
```



# Aliens Attack Version 4

## The draw-world Refinement

- `;; los scene → scene`  
`;; Purpose: To draw the given los in the given scene`  
`(define (draw-los a-los scn))`
- `(if (empty? a-los)`  
    `scn`  
    `(draw-shot (first a-los)`  
        `(draw-los (rest a-los) scn))))`
- `;; Sample expressions for draw-los`  
`(define ELOS-VAL E-SCENE)`  
`(define ALOS-VAL (draw-shot (first A-LOS)`  
    `(draw-los (rest A-LOS) E-SCENE)))`
- `;; Tests using sample computations for draw-los`  
`(check-expect (draw-los E-LOS E-SCENE) ELOS-VAL)`  
`(check-expect (draw-los A-LOS E-SCENE) ALOS-VAL)`  
`;; Tests using sample values for draw-los`  
`(check-expect (draw-los (cons (make-posn 14 8)`  
    `(cons (make-posn 3 2) '()))`  
    `E-SCENE2))`



Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The process-key Refinement

- The process-key function must be refined, because it builds worlds
- In addition, it must now always add new shots to the list of shots when the player presses the space bar

# Aliens Attack Version 4

## The process-key Refinement

- The `process-key` function must be refined, because it builds worlds
- In addition, it must now always add new shots to the list of shots when the player presses the space bar
- When the space bar is pressed a `posn` shot is always created and added to the world's list of shots
- This can be accomplished using `cons` given that it does not matter where the shot is added

# Aliens Attack Version 4

## The process-key Refinement

- `;; Sample expressions for process-key`  
`(define KEY-RVAL`  
    `(make-world`  
        `(move-rckt-right (world-rocket INIT-WORLD))`  
        `(world-aliens INIT-WORLD)`  
        `(world-dir INIT-WORLD)`  
        `(world-shots INIT-WORLD)))`  
`(define KEY-LVAL`  
    `(make-world`  
        `(move-rckt-left (world-rocket INIT-WORLD))`  
        `(world-aliens INIT-WORLD)`  
        `(world-dir INIT-WORLD)`  
        `(world-shots INIT-WORLD)))`  
`(define KEY-SVAL`  
    `(make-world`  
        `(world-rocket INIT-WORLD)`  
        `(world-aliens INIT-WORLD)`  
        `(world-dir INIT-WORLD)`  
        `(cons (process-shooting (world-rocket INIT-WORLD))`  
            `(world-shots INIT-WORLD))))`  
`(define KEY-SVAL2`  
    `(make-world`  
        `(world-rocket INIT-WORLD2)`  
        `(world-aliens INIT-WORLD2)`  
        `(world-dir INIT-WORLD2)`  
        `(cons (process-shooting (world-rocket INIT-WORLD2))`  
            `(world-shots INIT-WORLD2))))`  
`(define KEY-OVAL INIT-WORLD2)`

# Aliens Attack Version 4

## The process-key Refinement

- ```
;; world key → world
;; Purpose: Process a key event to return next world
(define (process-key a-world a-key)
```
- ```
; Sample expressions for process-key
(define KEY-RVAL
 (make-world
 (move-rckt-right (world-rocket INIT-WORLD))
 (world-aliens INIT-WORLD)
 (world-dir INIT-WORLD)
 (world-shots INIT-WORLD)))
(define KEY-LVAL
 (make-world
 (move-rckt-left (world-rocket INIT-WORLD))
 (world-aliens INIT-WORLD)
 (world-dir INIT-WORLD)
 (world-shots INIT-WORLD)))
(define KEY-SVAL
 (make-world
 (world-rocket INIT-WORLD)
 (world-aliens INIT-WORLD)
 (world-dir INIT-WORLD)
 (cons (process-shooting (world-rocket INIT-WORLD))
 (world-shots INIT-WORLD))))
(define KEY-SVAL2
 (make-world
 (world-rocket INIT-WORLD2)
 (world-aliens INIT-WORLD2)
 (world-dir INIT-WORLD2)
 (cons (process-shooting (world-rocket INIT-WORLD2))
 (world-shots INIT-WORLD2))))
(define KEY-OVAL INIT-WORLD2)
```

# Aliens Attack Version 4

## The process-key Refinement

- ```
; ; world key → world
;; Purpose: Process a key event to return next world
(define (process-key a-world a-key)
```

Aliens Attack Version 4

The process-key Refinement

- ```
; ; world key → world
; ; Purpose: Process a key event to return next world
(define (process-key a-world a-key)
```
- ```
; ; Tests using sample values for process-key
(check-expect
  (process-key (make-world (sub1 MAX-CHARS-HORIZONTAL) INIT-LOA
                            'right           E-LOS)
                "right")
  (make-world (sub1 MAX-CHARS-HORIZONTAL) INIT-LOA
              'right           E-LOS))
  (check-expect (process-key (make-world 0 INIT-LOA 'left E-LOS)
                             "left")
                (make-world 0 INIT-LOA 'left E-LOS))
  (check-expect (process-key (make-world 0 INIT-LOA 'left E-LOS)
                             "o")
                (make-world 0 INIT-LOA 'left E-LOS))
  (check-expect (process-key INIT-WORLD2 ";") INIT-WORLD2)
  (check-expect
    (process-key (make-world 0 INIT-LOA 'left E-LOS) " ")
    (make-world 0 INIT-LOA 'left (list (make-posn 0 MAX-IMG-Y))))
  (check-expect
    (process-key (make-world 0 INIT-LOA 'left (list SHOT2)) "left")
    (make-world 0 INIT-LOA 'left (list SHOT2))))
```

Aliens Attack Version 4

The process-key Refinement

- `; world key → world`
`; Purpose: Process a key event to return next world`
`(define (process-key a-world a-key)`

Aliens Attack Version 4

The process-key Refinement

- `; ; world key → world`
`; ; Purpose: Process a key event to return next world`
`(define (process-key a-world a-key)`
- `(cond [(key=? a-key "right")`
 `(make-world (move-rckt-right (world-rocket a-world))`
 `(world-aliens a-world)`
 `(world-dir a-world)`
 `(world-shots a-world))]`
`[(key=? a-key "left")`
 `(make-world (move-rckt-left (world-rocket a-world))`
 `(world-aliens a-world)`
 `(world-dir a-world)`
 `(world-shots a-world))]`
`[(key=? a-key " ")`
 `(make-world (world-rocket a-world)`
 `(world-aliens a-world)`
 `(world-dir a-world)`
 `(cons (process-shooting`
 `(world-rocket a-world))`
 `(world-shots a-world))))]`
`[else a-world]))`

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

The process-key Refinement

- Refining `process-key` is not done
- Important to implement all the changes made to the game

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

The process-key Refinement

- Refining `process-key` is not done
- Important to implement all the changes made to the game
- A shot must be added every time the space bar is pressed by the player

Aliens Attack Version 4

The process-key Refinement

- Refining `process-key` is not done
- Important to implement all the changes made to the game
- A shot must be added every time the space bar is pressed by the player
- This means that `process-shooting` must always return a `posn shot`
- In Aliens Attack 3, this function makes a decision to return either '`NO-SHOT` or a `posn shot`

Aliens Attack Version 4

The process-key Refinement

- Refining `process-key` is not done
- Important to implement all the changes made to the game
- A shot must be added every time the space bar is pressed by the player
- This means that `process-shooting` must always return a `posn shot`
- In Aliens Attack 3, this function makes a decision to return either '`NO-SHOT` or a `posn shot`
- The `image-x` value must be the given `rocket` value
- The `image-y` value must be `MAX-IMG-Y` to put the shot at the bottom at the scene

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

The process-key Refinement

- `;; Sample expressions for process-shooting`
`(define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))`
`(define PS-SHOT2-VAL (make-posn INIT-ROCKET2 MAX-IMG-Y))`

Aliens Attack Version 4

The process-key Refinement

- ```
;; shot rocket → shot
;; Purpose: To create a new shot
(define (process-shooting a-rocket)
```
- ```
;; Sample expressions for process-shooting
(define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))
(define PS-SHOT2-VAL (make-posn INIT-ROCKET2 MAX-IMG-Y))
```

Aliens Attack Version 4

The process-key Refinement

- ```
;; shot rocket → shot
;; Purpose: To create a new shot
(define (process-shooting a-rocket)
```
- ```
;; Sample expressions for process-shooting
(define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))
(define PS-SHOT2-VAL (make-posn INIT-ROCKET2 MAX-IMG-Y))
```
- ```
;; Tests using sample computations for process-shooting
(check-expect (process-shooting INIT-ROCKET) PS-SHOT-VAL1)
(check-expect (process-shooting INIT-ROCKET2) PS-SHOT2-VAL)

;; Tests using sample values for process-shooting
(check-expect (process-shooting 8) (make-posn 8 MAX-IMG-Y))
(check-expect (process-shooting 16) (make-posn 16 MAX-IMG-Y))
```

# Aliens Attack Version 4

## The process-key Refinement

- `;; shot rocket → shot`  
`;; Purpose: To create a new shot`  
`(define (process-shooting a-rocket)`
- `(make-posn a-rocket MAX-IMG-Y))`
- `;; Sample expressions for process-shooting`  
`(define PS-SHOT-VAL1 (make-posn INIT-ROCKET MAX-IMG-Y))`  
`(define PS-SHOT2-VAL (make-posn INIT-ROCKET2 MAX-IMG-Y))`
- `;; Tests using sample computations for process-shooting`  
`(check-expect (process-shooting INIT-ROCKET) PS-SHOT-VAL1)`  
`(check-expect (process-shooting INIT-ROCKET2) PS-SHOT2-VAL)`  
  
`;; Tests using sample values for process-shooting`  
`(check-expect (process-shooting 8) (make-posn 8 MAX-IMG-Y))`  
`(check-expect (process-shooting 16) (make-posn 16 MAX-IMG-Y))`

# Aliens Attack Version 4

## The process-tick Refinement

- In Aliens Attack 3, `process-tick` moves the alien, computes a new direction, and moves the shot
- What does `process-tick` have to do in Aliens Attack 4?

# Aliens Attack Version 4

## The process-tick Refinement

- In Aliens Attack 3, `process-tick` moves the alien, computes a new direction, and moves the shot
- What does `process-tick` have to do in Aliens Attack 4?
- Move a list of aliens **Designed in Section 76.1**
- Compute a new direction
- Move a list of shots **Designed in Section 76.2**

# Aliens Attack Version 4

## The process-tick Refinement

- In Aliens Attack 3, `process-tick` moves the alien, computes a new direction, and moves the shot
- What does `process-tick` have to do in Aliens Attack 4?
- Move a list of aliens **Designed in Section 76.1**
- Compute a new direction
- Move a list of shots **Designed in Section 76.2**
- Moving aliens and shots, however, is not enough
- As aliens and shots move they may hit each other and must be removed
- Removing hit aliens **Designed in Section 77.2**
- Removing hit shots **Designed in Section 76.3**

# Aliens Attack Version 4

## The process-tick Refinement

- In Aliens Attack 3, `process-tick` moves the alien, computes a new direction, and moves the shot
- What does `process-tick` have to do in Aliens Attack 4?
- Move a list of aliens [Designed in Section 76.1](#)
- Compute a new direction
- Move a list of shots [Designed in Section 76.2](#)
- Moving aliens and shots, however, is not enough
- As aliens and shots move they may hit each other and must be removed
- Removing hit aliens [Designed in Section 77.2](#)
- Removing hit shots [Designed in Section 76.3](#)
- What is required to compute a new direction?

# Aliens Attack Version 4

## The process-tick Refinement

- In Aliens Attack 3, `process-tick` moves the alien, computes a new direction, and moves the shot
- What does `process-tick` have to do in Aliens Attack 4?
- Move a list of aliens [Designed in Section 76.1](#)
- Compute a new direction
- Move a list of shots [Designed in Section 76.2](#)
- Moving aliens and shots, however, is not enough
- As aliens and shots move they may hit each other and must be removed
- Removing hit aliens [Designed in Section 77.2](#)
- Removing hit shots [Designed in Section 76.3](#)
- What is required to compute a new direction?
- In Aliens Attack 3, computing new direction uses the next (moved) alien and the current direction.
- Does it make sense to follow the same strategy using the (moved) list of aliens for the next world and the current direction?

# Aliens Attack Version 4

## The process-tick Refinement

- In Aliens Attack 3, `process-tick` moves the alien, computes a new direction, and moves the shot
- What does `process-tick` have to do in Aliens Attack 4?
- Move a list of aliens [Designed in Section 76.1](#)
- Compute a new direction
- Move a list of shots [Designed in Section 76.2](#)
- Moving aliens and shots, however, is not enough
- As aliens and shots move they may hit each other and must be removed
- Removing hit aliens [Designed in Section 77.2](#)
- Removing hit shots [Designed in Section 76.3](#)
- What is required to compute a new direction?
- In Aliens Attack 3, computing new direction uses the next (moved) alien and the current direction.
- Does it make sense to follow the same strategy using the (moved) list of aliens for the next world and the current direction?
- It seems perfectly reasonable because the new direction is for the next list of aliens

# Aliens Attack Version 4

## The process-tick Refinement

- ;; Sample expressions for process-tick  

```
(define AFTER-TICK-WORLD1
 (make-world
 (world-rocket INIT-WORLD)
 (remove-hit-aliens (move-loa (world-aliens INIT-WORLD) (world-dir INIT-WORLD))
 (move-los (world-shots INIT-WORLD)))
 (new-dir-after-tick
 (remove-hit-aliens
 (move-loa (world-aliens INIT-WORLD) (world-dir INIT-WORLD))
 (move-los (world-shots INIT-WORLD)))
 (world-dir INIT-WORLD))
 (remove-shots (move-los (world-shots INIT-WORLD))
 (move-loa (world-aliens INIT-WORLD) (world-dir INIT-WORLD)))))

(define AFTER-TICK-WORLD2
 (make-world
 (world-rocket INIT-WORLD2)
 (remove-hit-aliens (move-loa (world-aliens INIT-WORLD2) (world-dir INIT-WORLD2))
 (move-los (world-shots INIT-WORLD2)))
 (new-dir-after-tick
 (remove-hit-aliens
 (move-loa (world-aliens INIT-WORLD2) (world-dir INIT-WORLD2))
 (move-los (world-shots INIT-WORLD2)))
 (world-dir INIT-WORLD2))
 (remove-shots (move-los (world-shots INIT-WORLD2))
 (move-loa (world-aliens INIT-WORLD2) (world-dir INIT-WORLD2))))
```

# Aliens Attack Version 4

## The process-tick Refinement

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- `;; world → world` Purpose: Create a new world after a clock tick  
`(define (process-tick a-world)`

# Aliens Attack Version 4

## The process-tick Refinement

- `;; world → world Purpose: Create a new world after a clock tick`  
`(define (process-tick a-world)`
- `(make-world (world-rocket a-world)`  
`(remove-hit-aliens (move-loa (world-aliens a-world)`  
`(world-dir a-world))`  
`(move-los (world-shots a-world)))`  
`(new-dir-after-tick`  
`(remove-hit-aliens`  
`(move-loa (world-aliens a-world) (world-dir a-world)`  
`(move-los (world-shots a-world))))`  
`(world-dir a-world))`  
`(remove-shots (move-los (world-shots a-world))`  
`(move-loa (world-aliens a-world)`  
`(world-dir a-world))))`

# Aliens Attack Version 4

## The process-tick Refinement

- ;; Tests using sample values for process-tick
  - (check-expect (process-tick (make-world INIT-ROCKET (cons (make-posn 1 5) '()) 'left E-LOS)) (make-world INIT-ROCKET (cons (make-posn MIN-IMG-X 5) '()) 'down E-LOS)))
  - (check-expect (process-tick (make-world INIT-ROCKET2 (list (make-posn (- MAX-CHARS-HORIZONTAL 2) 10)) 'right (cons SHOT2 '())))) (make-world INIT-ROCKET2 (list (make-posn MAX-IMG-X 10)) 'down (cons (move-shot SHOT2) '()))))
  - (check-expect (process-tick (make-world INIT-ROCKET2 (cons (make-posn MAX-IMG-X 2) '()) 'down (cons (make-posn 15 6) '())))) (make-world INIT-ROCKET2 (cons (make-posn MAX-IMG-X 3) '()) 'left (cons (make-posn 15 5) '()))))
  - (check-expect (process-tick (make-world INIT-ROCKET2 (list (make-posn MIN-IMG-X 2)) 'down (list (make-posn 2 MIN-IMG-Y)))) (make-world INIT-ROCKET2 (list (make-posn MIN-IMG-X 3)) 'right '()))))

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The process-tick Refinement

- **new-dir-after-tick:** new direction depends on given direction

# Aliens Attack Version 4

## The process-tick Refinement

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- **new-dir-after-tick:** new direction depends on given direction
- If the given direction is down then the new direction may be right or left
- If the given direction is right then the new direction may be right or down
- If the given direction is left then the new direction may be left or down

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The process-tick Refinement

- **new-dir-after-tick:** new direction depends on given direction
- If the given direction is down then the new direction may be right or left
- If the given direction is right then the new direction may be right or down
- If the given direction is left then the new direction may be left or down
- This must be done for a loa

# Aliens Attack Version 4

## The process-tick Refinement

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- ; Sample expressions for new-dir-after-tick  
(define NEW-DIR-EDGE-LOA-DOWN (new-dir-after-down EDGE-LOA))  
(define NEW-DIR-EDGE-LOA2-DOWN (new-dir-after-down EDGE-LOA2))  
(define NEW-DIR-INIT-LOA-LEFT (new-dir-after-left INIT-LOA))  
(define NEW-DIR-EDGE-LOA-LEFT (new-dir-after-left EDGE-LOA))  
(define NEW-DIR-INIT-LOA-RIGHT (new-dir-after-right INIT-LOA))  
(define NEW-DIR-EDGE-LOA2-RIGHT (new-dir-after-right EDGE-LOA2))

# Aliens Attack Version 4

## The process-tick Refinement

- `;; loa dir → dir Purpose: Return new aliens direction  
(define (new-dir-after-tick a-loa old-dir)`
  
- `;; Sample expressions for new-dir-after-tick  
(define NEW-DIR-EDGE-LOA-DOWN (new-dir-after-down EDGE-LOA))  
(define NEW-DIR-EDGE-LOA2-DOWN (new-dir-after-down EDGE-LOA2))  
(define NEW-DIR-INIT-LOA-LEFT (new-dir-after-left INIT-LOA))  
(define NEW-DIR-EDGE-LOA-LEFT (new-dir-after-left EDGE-LOA))  
(define NEW-DIR-INIT-LOA-RIGHT (new-dir-after-right INIT-LOA))  
(define NEW-DIR-EDGE-LOA2-RIGHT (new-dir-after-right EDGE-LOA2))`

# Aliens Attack Version 4

## The process-tick Refinement

- `;; loa dir → dir Purpose: Return new aliens direction`  
`(define (new-dir-after-tick a-loa old-dir)`

- `;; Sample expressions for new-dir-after-tick`  
`(define NEW-DIR-EDGE-LOA-DOWN (new-dir-after-down EDGE-LOA))`  
`(define NEW-DIR-EDGE-LOA2-DOWN (new-dir-after-down EDGE-LOA2))`  
`(define NEW-DIR-INIT-LOA-LEFT (new-dir-after-left INIT-LOA))`  
`(define NEW-DIR-EDGE-LOA-LEFT (new-dir-after-left EDGE-LOA))`  
`(define NEW-DIR-INIT-LOA-RIGHT (new-dir-after-right INIT-LOA))`  
`(define NEW-DIR-EDGE-LOA2-RIGHT (new-dir-after-right EDGE-LOA2))`
- `;; Tests using sample computations for new-dir-after-tick`  
`(check-expect (new-dir-after-tick EDGE-LOA 'down) NEW-DIR-EDGE-LOA-DOWN)`  
`(check-expect (new-dir-after-tick EDGE-LOA2 'down) NEW-DIR-EDGE-LOA2-DOWN)`  
`(check-expect (new-dir-after-tick INIT-LOA 'left) NEW-DIR-INIT-LOA-LEFT)`  
`(check-expect (new-dir-after-tick EDGE-LOA 'left) NEW-DIR-EDGE-LOA-LEFT)`  
`(check-expect (new-dir-after-tick INIT-LOA 'right) NEW-DIR-INIT-LOA-RIGHT)`  
`(check-expect (new-dir-after-tick EDGE-LOA2 'right) NEW-DIR-EDGE-LOA2-RIGHT)`  
`;; Tests using sample values for new-dir-after-tick`  
`(check-expect (new-dir-after-tick (list (make-posn MIN-IMG-X 10)) 'down)`  
`'right)`  
`(check-expect (new-dir-after-tick (list (make-posn MAX-IMG-X 12)) 'down)`  
`'left)`  
`(check-expect (new-dir-after-tick (list (make-posn 10 10)) 'left)`  
`'left)`  
`(check-expect (new-dir-after-tick (list (make-posn MIN-IMG-X 15)) 'left)`  
`'down)`  
`(check-expect (new-dir-after-tick (list (make-posn 10 14)) 'right)`  
`'right)`  
`(check-expect (new-dir-after-tick (list (make-posn MAX-IMG-X 3)) 'right)`  
`'down)`

# Aliens Attack Version 4

## The process-tick Refinement

- `;; loa dir → dir` Purpose: Return new aliens direction  
`(define (new-dir-after-tick a-loa old-dir)`  
 `(cond [(eq? old-dir 'down) (new-dir-after-down a-loa)]`  
 `[(eq? old-dir 'left) (new-dir-after-left a-loa)]`  
 `[else (new-dir-after-right a-loa)]))`
- `;; Sample expressions for new-dir-after-tick`  
`(define NEW-DIR-EDGE-LOA-DOWN (new-dir-after-down EDGE-LOA))`  
`(define NEW-DIR-EDGE-LOA2-DOWN (new-dir-after-down EDGE-LOA2))`  
`(define NEW-DIR-INIT-LOA-LEFT (new-dir-after-left INIT-LOA))`  
`(define NEW-DIR-EDGE-LOA-LEFT (new-dir-after-left EDGE-LOA))`  
`(define NEW-DIR-INIT-LOA-RIGHT (new-dir-after-right INIT-LOA))`  
`(define NEW-DIR-EDGE-LOA2-RIGHT (new-dir-after-right EDGE-LOA2))`
- `;; Tests using sample computations for new-dir-after-tick`  
`(check-expect (new-dir-after-tick EDGE-LOA 'down) NEW-DIR-EDGE-LOA-DOWN)`  
`(check-expect (new-dir-after-tick EDGE-LOA2 'down) NEW-DIR-EDGE-LOA2-DOWN)`  
`(check-expect (new-dir-after-tick INIT-LOA 'left) NEW-DIR-INIT-LOA-LEFT)`  
`(check-expect (new-dir-after-tick EDGE-LOA 'left) NEW-DIR-EDGE-LOA-LEFT)`  
`(check-expect (new-dir-after-tick INIT-LOA 'right) NEW-DIR-INIT-LOA-RIGHT)`  
`(check-expect (new-dir-after-tick EDGE-LOA2 'right) NEW-DIR-EDGE-LOA2-RIGHT)`  
`;; Tests using sample values for new-dir-after-tick`  
`(check-expect (new-dir-after-tick (list (make-posn MIN-IMG-X 10)) 'down)`  
 `'right)`  
`(check-expect (new-dir-after-tick (list (make-posn MAX-IMG-X 12)) 'down)`  
 `'left)`  
`(check-expect (new-dir-after-tick (list (make-posn 10 10)) 'left)`  
 `'left)`  
`(check-expect (new-dir-after-tick (list (make-posn MIN-IMG-X 15)) 'left)`  
 `'down)`  
`(check-expect (new-dir-after-tick (list (make-posn 10 14)) 'right)`  
 `'right)`  
`(check-expect (new-dir-after-tick (list (make-posn MAX-IMG-X 3)) 'right)`  
 `'down))`

# Aliens Attack Version 4

## The process-tick Refinement

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- new-dir-after-down must decide for the given list of aliens if the next direction is right or left
- The needed conditional must determine if the given loa has an alien at the right edge or at the left edge
- The assumption is made that the given loa does not have aliens at both edges
- Without loss of generality, we choose to determine if the given loa contains an alien at the left edge
- We use any-alien-at-left-edge? designed in Section 74.1

# Aliens Attack Version 4

## The process-tick Refinement

- `; Sample expressions for new-dir-after-down  
(define AT-LEDGE-DOWN 'right)  
(define AT-REDGE-DOWN 'left)`

# Aliens Attack Version 4

## The process-tick Refinement

- ```
;; loa → direction
;; Purpose: Compute the direction of the given alien
;;           when previous direction is down
;; ASSUMPTION: The given loa does not have aliens at
;;           both edges
(define (new-dir-after-down a-loa)
```
- ```
;; Sample expressions for new-dir-after-down
(define AT-LEDGE-DOWN 'right)
(define AT-REDGE-DOWN 'left)
```

# Aliens Attack Version 4

## The process-tick Refinement

- ```
;; loa → direction
;; Purpose: Compute the direction of the given alien
;;           when previous direction is down
;; ASSUMPTION: The given loa does not have aliens at
;;           both edges
(define (new-dir-after-down a-loa)
```
- ```
(if (any-alien-at-left-edge? a-loa)
 'right
 'left))
```
- ```
; Sample expressions for new-dir-after-down
(define AT-LEDGE-DOWN 'right)
(define AT-REDGE-DOWN 'left)
```
- ```
; Tests using sample computations for new-dir-after-down
(check-expect (new-dir-after-down EDGE-LOA2) AT-REDGE-DOWN)
(check-expect (new-dir-after-down EDGE-LOA) AT-LEDGE-DOWN)
```
  
- ```
; Tests using sample values for new-dir-after-down
(check-expect (new-dir-after-down
                (list (make-posn MIN-IMG-X 4)))
              'right)
(check-expect (new-dir-after-down
                (list (make-posn MAX-IMG-X 9)))
              'left)
```

Aliens Attack Version 4

The process-tick Refinement

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- new-dir-after-left must decide for the given list of aliens if the next direction is down or left
- The needed conditional must determine if the given loa has an alien at the left edge
- If so, the next direction is down
- Otherwise, it is left
- We use any-alien-at-left-edge? designed in Section 74.1

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

The process-tick Refinement

- ```
;; Sample expressions for new-dir-after-left
(define AT-LEDGE 'down)
(define NOT-AT-LEDGE 'left)
```

# Aliens Attack Version 4

## The process-tick Refinement

- ```
;; alien → direction
;; Purpose: Compute the direction of the given alien
;;           when previous direction is left
(define (new-dir-after-left a-loa)
```
- ```
;; Sample expressions for new-dir-after-left
(define AT-LEDGE 'down)
(define NOT-AT-LEDGE 'left)
```

# Aliens Attack Version 4

## The process-tick Refinement

- ```
;; alien → direction
;; Purpose: Compute the direction of the given alien
;;           when previous direction is left
(define (new-dir-after-left a-loa)
```
- ```
;; Sample expressions for new-dir-after-left
(define AT-LEDGE 'down)
(define NOT-AT-LEDGE 'left)
```
- ```
;; Tests using sample computations for new-dir-after-left
(check-expect (new-dir-after-left EDGE-LOA) AT-LEDGE)
(check-expect (new-dir-after-left INIT-LOA) NOT-AT-LEDGE)

;; Tests using sample values for new-dir-after-left
(check-expect (new-dir-after-left (list RIGHT-EDGE-ALIEN))
              'left)
```

Aliens Attack Version 4

The process-tick Refinement

- ```
;; alien → direction
;; Purpose: Compute the direction of the given alien
;; when previous direction is left
(define (new-dir-after-left a-loa)

 (if (any-alien-at-left-edge? a-loa)
 'down
 'left))

;; Sample expressions for new-dir-after-left
(define AT-LEDGE 'down)
(define NOT-AT-LEDGE 'left)

;; Tests using sample computations for new-dir-after-left
(check-expect (new-dir-after-left EDGE-LOA) AT-LEDGE)
(check-expect (new-dir-after-left INIT-LOA) NOT-AT-LEDGE)

;; Tests using sample values for new-dir-after-left
(check-expect (new-dir-after-left (list RIGHT-EDGE-ALIEN))
 'left)
```

# Aliens Attack Version 4

## The process-tick Refinement

- new-dir-after-right must decide for the given list of aliens if the next direction is down or right
- The needed conditional must determine if the given loa has an alien at the right edge
- If so, the next direction is down
- Otherwise, it is right
- We use any-alien-at-right-edge? designed in Section 74.2

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The process-tick Refinement

- ```
;; Sample expressions for new-dir-after-right
(define AT-EDGE      'down)
(define NOT-AT-EDGE 'right)
```

Aliens Attack Version 4

The process-tick Refinement

- ```
;; loa → direction
;; Purpose: Compute the direction of the given loa
;; when previous direction is right
(define (new-dir-after-right a-loa)
```
- ```
;; Sample expressions for new-dir-after-right
(define AT-EDGE      'down)
(define NOT-AT-EDGE 'right)
```

Aliens Attack Version 4

The process-tick Refinement

- ;; loa → direction
;; Purpose: Compute the direction of the given loa
;; when previous direction is right
(define (new-dir-after-right a-loa)

•
;; Sample expressions for new-dir-after-right
(define AT-EDGE 'down)
(define NOT-AT-EDGE 'right)

•
;; Tests using sample computations for new-dir-after-right
(check-expect (new-dir-after-right EDGE-LOA2)
 AT-EDGE)
(check-expect (new-dir-after-right INIT-LOA)
 NOT-AT-EDGE)

;; Tests using sample values for new-dir-after-right
(check-expect (new-dir-after-right (list LEFT-EDGE-ALIEN))
 'right)

Aliens Attack Version 4

The process-tick Refinement

- ```
;; loa → direction
;; Purpose: Compute the direction of the given loa
;; when previous direction is right
(define (new-dir-after-right a-loa)
```
- ```
(if (any-alien-at-right-edge? a-loa)
      'down
      'right))
```
- ```
; Sample expressions for new-dir-after-right
(define AT-EDGE 'down)
(define NOT-AT-EDGE 'right)
```
- ```
; Tests using sample computations for new-dir-after-right
(check-expect (new-dir-after-right EDGE-LOA2)
              AT-EDGE)
(check-expect (new-dir-after-right INIT-LOA)
              NOT-AT-EDGE)
```
- ```
; Tests using sample values for new-dir-after-right
(check-expect (new-dir-after-right (list LEFT-EDGE-ALIEN))
 'right)
```

# Aliens Attack Version 4

## The game-over? Refinement

- When should the game be over?
- When does the player lose the game?

# Aliens Attack Version 4

## The game-over? Refinement

- When should the game be over?
- When does the player lose the game?
- The player loses when any alien reaches earth **Designed in Section 74.3**
- When does the player win the game?

# Aliens Attack Version 4

## The game-over? Refinement

- When should the game be over?
- When does the player lose the game?
- The player loses when any alien reaches earth **Designed in Section 74.3**
- When does the player win the game?
- The player wins when there are no more aliens left

# Aliens Attack Version 4

## The game-over? Refinement

- (define WORLD4 (make-world 4 '() 'left '()))

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The game-over? Refinement

- (define WORLD4 (make-world 4 '() 'left '()))

- ; Sample expressions for game-over?  
(define GAME-OVER1 (or (any-alien-reached-earth? (world-aliens INIT-WORLD2))  
                          (not (any-aliens-alive? (world-aliens INIT-WORLD2)))))  
  
(define GAME-OVER2 (or (any-alien-reached-earth? (world-aliens WORLD4))  
                          (not (any-aliens-alive? (world-aliens WORLD4)))))  
  
(define GAME-NOT-OVER (or (any-alien-reached-earth? (world-aliens INIT-WORLD))  
                          (not (any-aliens-alive? (world-aliens INIT-WORLD))))))

# Aliens Attack Version 4

## The game-over? Refinement

- (define WORLD4 (make-world 4 '() 'left '()))
- ;; world → Boolean Purpose: Detect if the game is over  
(define (game-over? a-world)
  - ;; Sample expressions for game-over?  
(define GAME-OVER1 (or (any-alien-reached-earth? (world-aliens INIT-WORLD2))  
                                  (not (any-aliens-alive? (world-aliens INIT-WORLD2)))))  
  
(define GAME-OVER2 (or (any-alien-reached-earth? (world-aliens WORLD4))  
                                  (not (any-aliens-alive? (world-aliens WORLD4)))))  
  
(define GAME-NOT-OVER (or (any-alien-reached-earth? (world-aliens INIT-WORLD))  
                                  (not (any-aliens-alive? (world-aliens INIT-WORLD))))))

# Aliens Attack Version 4

## The game-over? Refinement

- ```
(define WORLD4 (make-world 4 '() 'left '()))
```
- ```
; world → Boolean Purpose: Detect if the game is over
(define (game-over? a-world)
```
  
- ```
; Sample expressions for game-over?
(define GAME-OVER1 (or (any-alien-reached-earth? (world-aliens INIT-WORLD2))
                           (not (any-aliens-alive? (world-aliens INIT-WORLD2)))))

(define GAME-OVER2 (or (any-alien-reached-earth? (world-aliens WORLD4))
                           (not (any-aliens-alive? (world-aliens WORLD4)))))

(define GAME-NOT-OVER (or (any-alien-reached-earth? (world-aliens INIT-WORLD))
                           (not (any-aliens-alive? (world-aliens INIT-WORLD))))))
```
- ```
; Tests using sample computations for game-over?
(check-expect (game-over? INIT-WORLD2) GAME-OVER1)
(check-expect (game-over? WORLD4) GAME-OVER2)
(check-expect (game-over? INIT-WORLD) GAME-NOT-OVER)

; Tests using sample values for game-over?
(check-expect
 (game-over? (make-world 8 (list (make-posn 0 3)) 'right NO-SHOT))
 #false)
(check-expect
 (game-over?
 (make-world 8 (list (make-posn 0 MAX-IMG-Y)) 'right (list (make-posn 12 11))))
 #true)
(check-expect
 (game-over? (make-world 8 (list (make-posn 0 5)) 'right (list (make-posn 0 5))))
 #false)
```

# Aliens Attack Version 4

## The game-over? Refinement

- (define WORLD4 (make-world 4 '() 'left '()))
- ; world → Boolean Purpose: Detect if the game is over  
(define (game-over? a-world)  
 (or (any-alien-reached-earth? (world-aliens a-world))  
 (not (any-aliens-alive? (world-aliens a-world))))))
- ; Sample expressions for game-over?  
(define GAME-OVER1 (or (any-alien-reached-earth? (world-aliens INIT-WORLD2))  
 (not (any-aliens-alive? (world-aliens INIT-WORLD2)))))
- (define GAME-OVER2 (or (any-alien-reached-earth? (world-aliens WORLD4))  
 (not (any-aliens-alive? (world-aliens WORLD4)))))
- (define GAME-NOT-OVER (or (any-alien-reached-earth? (world-aliens INIT-WORLD))  
 (not (any-aliens-alive? (world-aliens INIT-WORLD)))))
- ; Tests using sample computations for game-over?  
(check-expect (game-over? INIT-WORLD2) GAME-OVER1)  
(check-expect (game-over? WORLD4) GAME-OVER2)  
(check-expect (game-over? INIT-WORLD) GAME-NOT-OVER)  
;; Tests using sample values for game-over?  
(check-expect  
 (game-over? (make-world 8 (list (make-posn 0 3)) 'right NO-SHOT))  
 #false)  
(check-expect  
 (game-over?  
 (make-world 8 (list (make-posn 0 MAX-IMG-Y)) 'right (list (make-posn 12 11))))  
 #true)  
(check-expect  
 (game-over? (make-world 8 (list (make-posn 0 5)) 'right (list (make-posn 0 5))))  
 #false)

# Aliens Attack Version 4

## The game-over? Refinement

- any-aliens-alive? does not have to process the given loa
- It needs to test if the given loa is not empty

# Aliens Attack Version 4

## The game-over? Refinement

- any-aliens-alive? does not have to process the given loa
- It needs to test if the given loa is not empty

- ```
;; Sample expressions for any-aliens-alive?
(define NOT-ALIVE-VAL (cons? E-LOA))
(define ALIVE-VAL      (cons? INIT-LOA))
```

Aliens Attack Version 4

The game-over? Refinement

- any-aliens-alive? does not have to process the given loa
- It needs to test if the given loa is not empty
- ```
;; loa Boolean → Boolean
;; Purpose: Determine if there is a posn alien in the given loa
(define (any-aliens-alive? a-loa)
```
- ```
;; Sample expressions for any-aliens-alive?
(define NOT-ALIVE-VAL (cons? E-LOA))
(define ALIVE-VAL      (cons? INIT-LOA))
```

Aliens Attack Version 4

The game-over? Refinement

- any-aliens-alive? does not have to process the given loa
- It needs to test if the given loa is not empty
- ```
;; loa Boolean → Boolean
;; Purpose: Determine if there is a posn alien in the given loa
(define (any-aliens-alive? a-loa)
```
- ```
;; Sample expressions for any-aliens-alive?
(define NOT-ALIVE-VAL (cons? E-LOA))
(define ALIVE-VAL      (cons? INIT-LOA))
```
- ```
;; Tests using sample computations any-aliens-alive?
(check-expect (any-aliens-alive? E-LOA) NOT-ALIVE-VAL)
(check-expect (any-aliens-alive? INIT-LOA) ALIVE-VAL)
```
- ```
;; Tests using sample values for any-aliens-alive?
(check-expect (any-aliens-alive? '()) #false)
(check-expect (any-aliens-alive? (list (make-posn 9 2)
                                         (make-posn 6 4)))
                         #true)
```

Aliens Attack Version 4

The game-over? Refinement

- any-aliens-alive? does not have to process the given loa
- It needs to test if the given loa is not empty
- ```
;; loa Boolean → Boolean
;; Purpose: Determine if there is a posn alien in the given loa
(define (any-aliens-alive? a-loa)
```
- ```
(cons? a-loa))
```
- ```
;; Sample expressions for any-aliens-alive?
(define NOT-ALIVE-VAL (cons? E-LOA))
(define ALIVE-VAL (cons? INIT-LOA))
```
- ```
;; Tests using sample computations any-aliens-alive?
(check-expect (any-aliens-alive? E-LOA)    NOT-ALIVE-VAL)
(check-expect (any-aliens-alive? INIT-LOA)  ALIVE-VAL)
```
- ```
;; Tests using sample values for any-aliens-alive?
(check-expect (any-aliens-alive? '()) #false)
(check-expect (any-aliens-alive? (list (make-posn 9 2)
 (make-posn 6 4)))
 #true)
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The draw-last-world Refinement

- Recall that this function is used only when game-over? returns #true
- How do we determine if the player has won or lost?

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Aliens Attack Version 4

## The draw-last-world Refinement

- Recall that this function is used only when game-over? returns #true
- How do we determine if the player has won or lost?
- We may use any-alien-reached-earth? and any-aliens-alive?

# Aliens Attack Version 4

## The draw-last-world Refinement

- ;; Sample Instance of (final) world  

```
(define FWORLD1 (make-world 13
 (list (make-posn 0 MAX-IMG-Y))
 'right
 E-LOS))
(define FWORLD2 (make-world 7
 (list (make-posn 19 MAX-IMG-Y))
 'left
 (list (make-posn 2 4))))
(define FWORLD3 (make-world 7 '() 'left (list (make-posn 19 3))))
```

# Aliens Attack Version 4

## The draw-last-world Refinement

- ;; Sample Instance of (final) world  

```
(define FWORLD1 (make-world 13
 (list (make-posn 0 MAX-IMG-Y))
 'right
 E-LOS))
(define FWORLD2 (make-world 7
 (list (make-posn 19 MAX-IMG-Y))
 'left
 (list (make-posn 2 4))))
(define FWORLD3 (make-world 7 '() 'left (list (make-posn 19 3))))
```
- ;; Sample expressions for draw-last-world  

```
(define FWORLD1-VAL (place-image
 (text "EARTH WAS CONQUERED!" 36 'red)
 (/ E-SCENE-W 2)
 (/ E-SCENE-H 4)
 (draw-world FWORLD1)))
(define FWORLD2-VAL (place-image
 (text "EARTH WAS CONQUERED!" 36 'red)
 (/ E-SCENE-W 2)
 (/ E-SCENE-H 4)
 (draw-world FWORLD2)))
(define FWORLD3-VAL (place-image
 (text "EARTH WAS SAVED!" 36 'green)
 (/ E-SCENE-W 2)
 (/ E-SCENE-H 4)
 (draw-world FWORLD3)))
```

# Aliens Attack Version 4

## The draw-last-world Refinement

- ```
;; world → scene throws error
;; Purpose: To draw the game's final scene
(define (draw-last-world a-world)
  (cond [(any-alien-reached-earth? (world-aliens a-world))
          (place-image (text "EARTH WAS CONQUERED!" 36 'red)
                      (/ E-SCENE-W 2)
                      (/ E-SCENE-H 4)
                      (draw-world a-world))]
        [(not (any-aliens-alive? (world-aliens a-world)))
          (place-image (text "EARTH WAS SAVED!" 36 'green)
                      (/ E-SCENE-W 2)
                      (/ E-SCENE-H 4)
                      (draw-world a-world))]
        [else
         (error (format
                  "draw-last-world: Given world has ~s aliens and
                  none have reached earth."
                  (length (world-aliens a-world))))])
```

Aliens Attack Version 4

The draw-last-world Refinement

- ;; Tests using sample computations for draw-last-world
(check-expect (draw-last-world FWORLD1) FWORLD1-VAL)
(check-expect (draw-last-world FWORLD2) FWORLD2-VAL)
(check-expect (draw-last-world FWORLD3) FWORLD3-VAL)

;; Tests using sample values for draw-last-world
(check-expect (draw-last-world (make-world 10 '() 'left '()))



```
(check-error
  (draw-last-world
    (make-world 10
      (list (make-posn 3 3) (make-posn 7 8))
      'right
      (list (make-posn 2 3))))
    "draw-last-world: Given world has 2 aliens and none have
    reached earth.')
```

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- After playing the game several, have noticed that there is a bug?
- This is a bug that does not always manifests itself
- Define the following world:

```
(define BUG-WORLD
  (make-world
    10
    (list (make-posn 4 6) (make-posn 10 6) (make-posn 11 6)
          (make-posn 12 6) (make-posn 10 7) (make-posn 11 7)
          (make-posn 13 7) (make-posn 15 8))
    'left
    (list (make-posn 0 12))))
```

- There is an alien that is farther left the aliens are moving left

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- After playing the game several, have noticed that there is a bug?
- This is a bug that does not always manifests itself
- Define the following world:

```
(define BUG-WORLD
  (make-world
    10
    (list (make-posn 4 6) (make-posn 10 6) (make-posn 11 6)
          (make-posn 12 6) (make-posn 10 7) (make-posn 11 7)
          (make-posn 13 7) (make-posn 15 8)))
    'left
    (list (make-posn 0 12))))
```

- There is an alien that is farther left the aliens are moving left
- ;; string → world

```
; Purpose: To run the game
(define (run a-name)
  (big-bang BUG-WORLD
    [on-draw draw-world]
    [name a-name]
    [on-key process-key]
    [on-tick process-tick TICK-RATE]
    [stop-when game-over? draw-last-world]))
```

- Run the game several times and see if you can notice the bug
- What do you notice about the how the aliens move?

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Let's closely analyze what happens
- We shall focus on the first alien, the only shot, and the direction

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Let's closely analyze what happens
- We shall focus on the first alien, the only shot, and the direction
- This table captures how these values change as the clock ticks:

TICK	ALIEN	SHOT	DIRECTION
0	(make-posn 4 6)	(make-posn 0 12)	'left
1	(make-posn 3 6)	(make-posn 0 11)	'left
2	(make-posn 2 6)	(make-posn 0 10)	'left
3	(make-posn 1 6)	(make-posn 0 9)	'left
4	(make-posn 0 6)	(make-posn 0 8)	'left
5	(make-posn 0 7)	(make-posn 0 7)	'down

- At tick 4 the alien is at the left edge and the direction changes to down
- At tick 5 the alien has moved down and the shot has moved up hitting each other

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Let's closely analyze what happens
- We shall focus on the first alien, the only shot, and the direction
- This table captures how these values change as the clock ticks:

TICK	ALIEN	SHOT	DIRECTION
0	(make-posn 4 6)	(make-posn 0 12)	'left
1	(make-posn 3 6)	(make-posn 0 11)	'left
2	(make-posn 2 6)	(make-posn 0 10)	'left
3	(make-posn 1 6)	(make-posn 0 9)	'left
4	(make-posn 0 6)	(make-posn 0 8)	'left
5	(make-posn 0 7)	(make-posn 0 7)	'down

- At tick 4 the alien is at the left edge and the direction changes to down
- At tick 5 the alien has moved down and the shot has moved up hitting each other
- `process-tick` removes both from the world
- The new direction is computed using the list of aliens that does not contain the hit alien and 'down' by calling `new-dir-after-down`

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Let's closely analyze what happens
- We shall focus on the first alien, the only shot, and the direction
- This table captures how these values change as the clock ticks:

TICK	ALIEN	SHOT	DIRECTION
0	(make-posn 4 6)	(make-posn 0 12)	'left
1	(make-posn 3 6)	(make-posn 0 11)	'left
2	(make-posn 2 6)	(make-posn 0 10)	'left
3	(make-posn 1 6)	(make-posn 0 9)	'left
4	(make-posn 0 6)	(make-posn 0 8)	'left
5	(make-posn 0 7)	(make-posn 0 7)	'down

- At tick 4 the alien is at the left edge and the direction changes to down
- At tick 5 the alien has moved down and the shot has moved up hitting each other
- `process-tick` removes both from the world
- The new direction is computed using the list of aliens that does not contain the hit alien and 'down' by calling `new-dir-after-down`
- Tests if any alien is at the left edge: #false and the function returns 'left'
- The aliens go back to moving left when they should move right

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Let's closely analyze what happens
- We shall focus on the first alien, the only shot, and the direction
- This table captures how these values change as the clock ticks:

TICK	ALIEN	SHOT	DIRECTION
0	(make-posn 4 6)	(make-posn 0 12)	'left
1	(make-posn 3 6)	(make-posn 0 11)	'left
2	(make-posn 2 6)	(make-posn 0 10)	'left
3	(make-posn 1 6)	(make-posn 0 9)	'left
4	(make-posn 0 6)	(make-posn 0 8)	'left
5	(make-posn 0 7)	(make-posn 0 7)	'down

- At tick 4 the alien is at the left edge and the direction changes to down
- At tick 5 the alien has moved down and the shot has moved up hitting each other
- `process-tick` removes both from the world
- The new direction is computed using the list of aliens that does not contain the hit alien and 'down' by calling `new-dir-after-down`
- Tests if any alien is at the left edge: #false and the function returns 'left'
- The aliens go back to moving left when they should move right
- We were not careful enough in our original problem analysis: not reasonable to compute the new direction using the moved and filtered list of aliens

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Let's closely analyze what happens
- We shall focus on the first alien, the only shot, and the direction
- This table captures how these values change as the clock ticks:

TICK	ALIEN	SHOT	DIRECTION
0	(make-posn 4 6)	(make-posn 0 12)	'left
1	(make-posn 3 6)	(make-posn 0 11)	'left
2	(make-posn 2 6)	(make-posn 0 10)	'left
3	(make-posn 1 6)	(make-posn 0 9)	'left
4	(make-posn 0 6)	(make-posn 0 8)	'left
5	(make-posn 0 7)	(make-posn 0 7)	'down

- At tick 4 the alien is at the left edge and the direction changes to down
- At tick 5 the alien has moved down and the shot has moved up hitting each other
- `process-tick` removes both from the world
- The new direction is computed using the list of aliens that does not contain the hit alien and 'down' by calling `new-dir-after-down`
- Tests if any alien is at the left edge: #false and the function returns 'left'
- The aliens go back to moving left when they should move right
- We were not careful enough in our original problem analysis: not reasonable to compute the new direction using the moved and filtered list of aliens
- Despite hundreds of tests passing there is still a bug in the program
- Lesson: Tests do not guarantee that a program is correct

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Must revisit our problem analysis to fix the bug

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Must revisit our problem analysis to fix the bug
- Problem: hit alien is removed before computing the new direction

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Must revisit our problem analysis to fix the bug
- Problem: hit alien is removed before computing the new direction
- If the hit alien is not removed then `new-dir-after-down's` returns correct direction

Aliens Attack Version 4

A Bug Despite Hundreds of Tests Passing

- Must revisit our problem analysis to fix the bug
- Problem: hit alien is removed before computing the new direction
- If the hit alien is not removed then `new-dir-after-down's` returns correct direction
- Refine `process-tick` to be:

```
;; world → world
;; Purpose: Create a new world after a clock tick
(define (process-tick a-world)
  (make-world (world-rocket a-world)
              (remove-hit-aliens
                (move-loa (world-aliens a-world)
                          (world-dir a-world))
                (move-los (world-shots a-world)))
              (new-dir-after-tick
                (move-loa (world-aliens a-world)
                          (world-dir a-world))
                (world-dir a-world)))
              (remove-shots
                (move-los (world-shots a-world))
                (move-loa (world-aliens a-world)
                          (world-dir a-world))))))
```

Aliens Attack Version 4

Midterm Exam

- Problem: 180
- Work in groups
- Filename:
`<lastname>-<firstname>-...<lastname>-<firstname>-midterm.rkt`
- Due: 1 week from today
- Follow all the steps of the design recipe
- Have fun and be creative!

Binary Trees

- We have explored data of arbitrary size whose definitions only contain one selfreference:

A list of X is either:

1. '()
2. (cons X (listof X))

A natural number is either:

1. 0
2. (add1 natnum)

Binary Trees

- We have explored data of arbitrary size whose definitions only contain one selfreference:

A list of X is either:

1. '()
2. (cons X (listof X))

A natural number is either:

1. 0
2. (add1 natnum)

- There are, however, many data types that contain more than one selfreference

Binary Trees

- We have explored data of arbitrary size whose definitions only contain one selfreference:

A list of X is either:	A natural number is either:
1. '()	1. 0
2. (cons X (listof X))	2. (add1 natnum)
- There are, however, many data types that contain more than one selfreference
- Consider representing the results of an arbitrary number of coin flips

Binary Trees

- We have explored data of arbitrary size whose definitions only contain one selfreference:

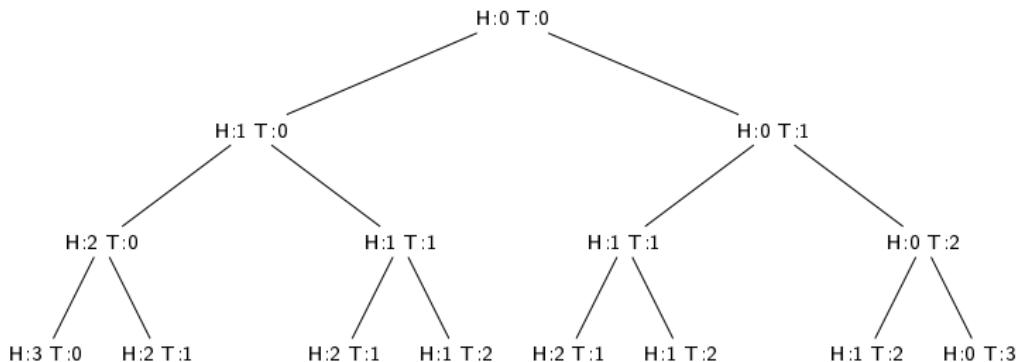
A list of X is either:

- '()
- (cons X (listof X))

A natural number is either:

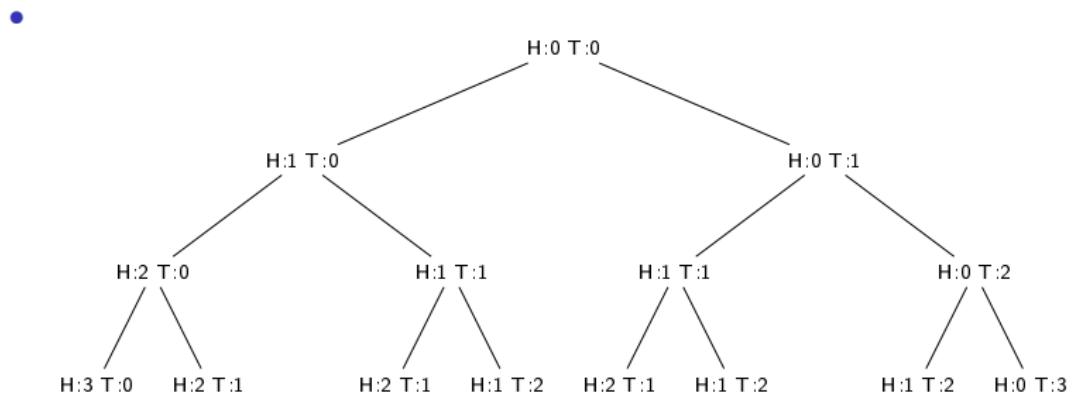
- 0
- (add1 natnum)

- There are, however, many data types that contain more than one selfreference
- Consider representing the results of an arbitrary number of coin flips
-



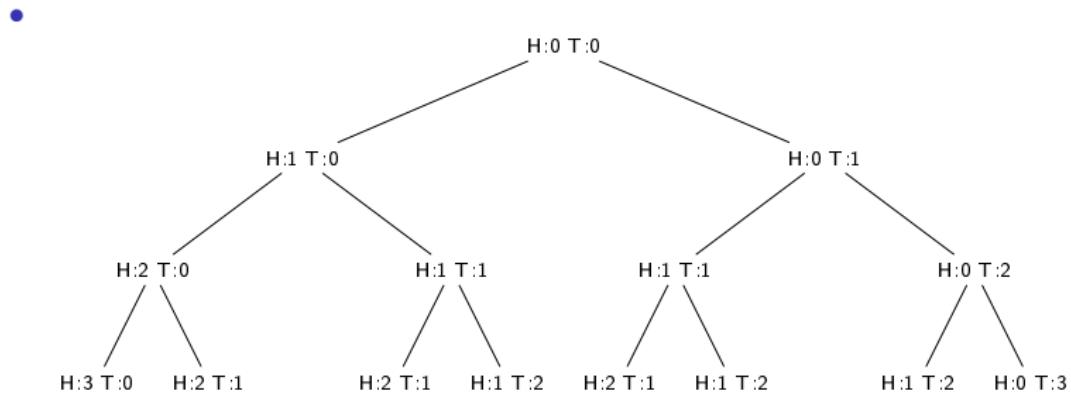
- At the top level, 0 heads and 0 tails have been flip
- Going left represents that a head is flipped and going right represents that a tail is flipped as you traverse down to the next level

Binary Trees



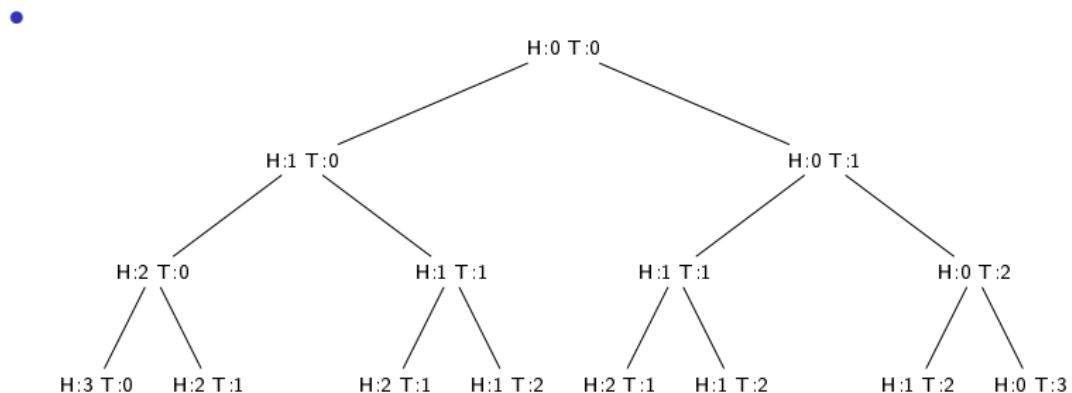
- The data representation is called a *binary tree*

Binary Trees



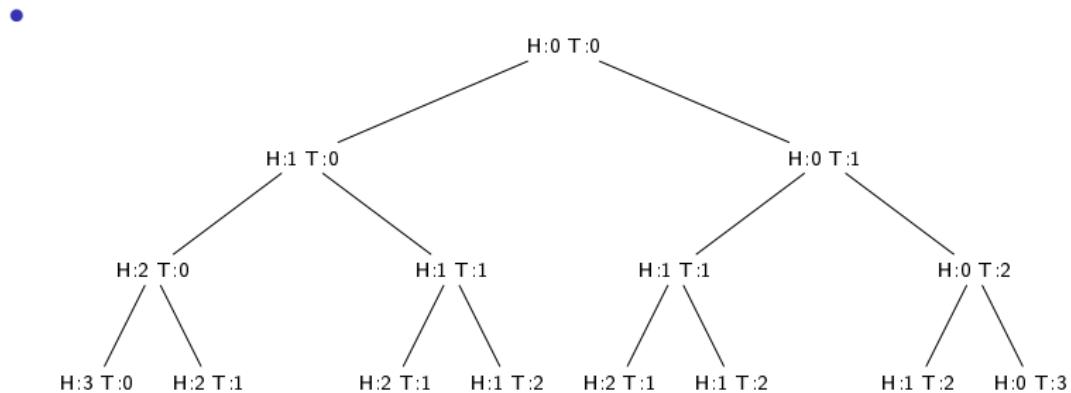
- The data representation is called a *binary tree*
- A binary tree may be empty
- A non-empty binary tree is made up of 1 or more nodes

Binary Trees



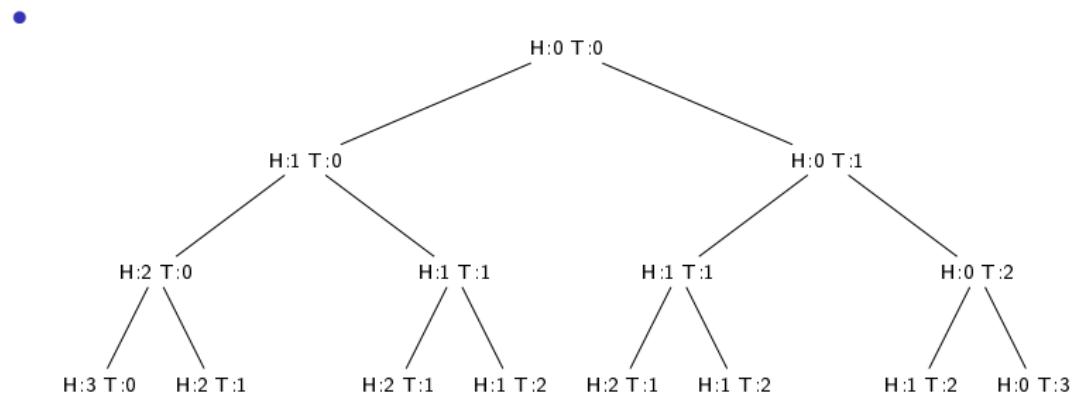
- The data representation is called a *binary tree*
- A binary tree may be empty
- A non-empty binary tree is made up of 1 or more nodes
- A node has data, like the number of heads and tails flipped, and at most two children

Binary Trees



- The data representation is called a *binary tree*
- A binary tree may be empty
- A non-empty binary tree is made up of 1 or more nodes
- A node has data, like the number of heads and tails flipped, and at most two children
- The parent node is connected to its children by an *edge*: *left child* and *right child*
- The top node of a binary tree is called the *root*
- A node that does not have any children is called a *leaf*

Binary Trees



- The data representation is called a *binary tree*
- A binary tree may be empty
- A non-empty binary tree is made up of 1 or more nodes
- A node has data, like the number of heads and tails flipped, and at most two children
- The parent node is connected to its children by an `edge`: `left child` and `right child`
- The top node of a binary tree is called the `root`
- A node that does not have any children is called a `leaf`
- Binary trees are commonly used to efficiently access data contained in nodes and to represent data with a bifurcating structure—a structure where placement of a node (e.g., left or right) is part of the information being represented

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

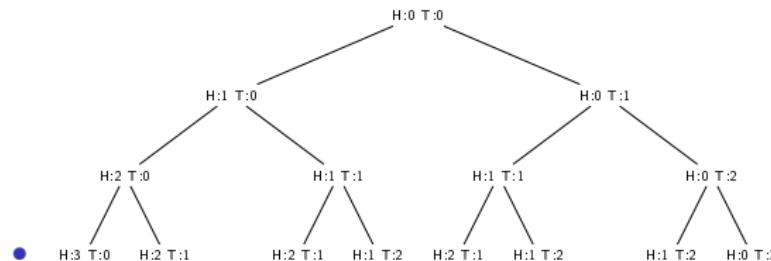
Binary Tree Data Definition

- Like a list, a binary tree may contain any kind of data
- We are free to define that type of data each node contains

Binary Trees

Binary Tree Data Definition

- Like a list, a binary tree may contain any kind of data
- We are free to define that type of data each node contains

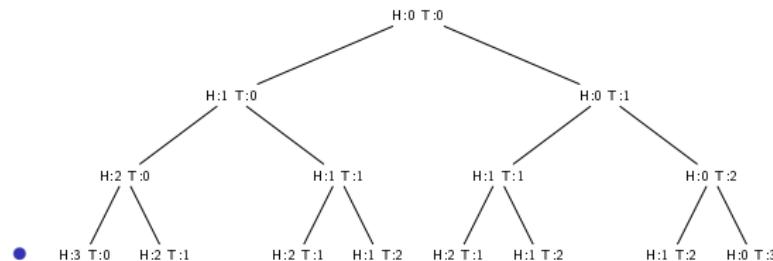


- Binary tree of coin-tally, where a coin-tally is a structure with two numbers

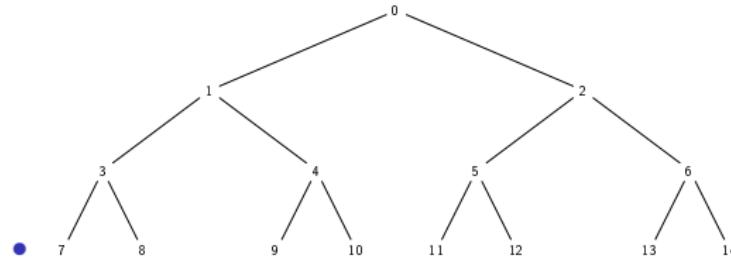
Binary Trees

Binary Tree Data Definition

- Like a list, a binary tree may contain any kind of data
- We are free to define that type of data each node contains



- Binary tree of coin-tally, where a coin-tally is a structure with two numbers



- Binary tree of natnum

Binary Trees

Binary Tree Data Definition

- `;; A cointally is a structure: (make-cointally natnum natnum)
(define-struct cointally (heads tails))`

`;; A binary tree of coin tallies, (btotf cointally), is either:
;; 1. ()
;; 2. (list cointally (btotf cointally) (btotf cointally))`

Binary Trees

Binary Tree Data Definition

- `;; A cointally is a structure: (make-cointally natnum natnum)`
`(define-struct cointally (heads tails))`

`;; A binary tree of coin tallies, (btot cointally), is either:`
`;; 1. ()`
`;; 2. (list cointally (btot cointally) (btot cointally))`
- `;; A binary tree of natural numbers, (btot natnum), is either:`
`;; 1. ()`
`;; 2. (list natnum (btot natnum) (btot natnum))`

Binary Trees

Binary Tree Data Definition

- `;; A cointally is a structure: (make-cointally natnum natnum)`
`(define-struct cointally (heads tails))`

`;; A binary tree of coin tallies, (btot cointally), is either:`
`;; 1. ()`
`;; 2. (list cointally (btot cointally) (btot cointally))`
- `;; A binary tree of natural numbers, (btot natnum), is either:`
`;; 1. ()`
`;; 2. (list natnum (btot natnum) (btot natnum))`
- Abstracting over the data definitions above yields the following generic data definition:

`;; A binary tree of X, (btot X), is either:`
`;; 1. ()`
`;; 2. (list X (btot X) (btot X))`

Binary Trees

Binary Tree Data Definition

- ```
;; Sample instances of btx
(define E-BTX '())
(define BTX2 ...)

;; (btot X) ... → ...
;; Purpose: ...
(define (f-on-btx a-btx ...)
 (if (empty? a-btx)
 ...
 ... (f-on-X (first a-btx))
 (f-on-btx (second a-btx))
 (f-on-btx (thrid a-btx))...)))

;; Sample expressions for f-on-btx
(define E-BTX-VAL ...)
(define BTX2-VAL ...)

;; Tests using sample computations for f-on-btx
(check-expect (f-on-btx E-BTX ...) E-BTX-VAL)
(check-expect (f-on-btx BTX2 ...) BTX2-VAL) ...

;; Tests using sample values for f-on-btx
(check-expect (f-on-btx ...) ...)
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

## Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## Traversing a Binary Tree

- There are different ways a binary tree may be traversed

# Binary Trees

## Traversing a Binary Tree

- There are different ways a binary tree may be traversed
- Definition template for a function on a (`btof X`):
  - a call to a function on a `X`
  - a call to process the left subtree
  - a call to process the right subtree
- The template does not prescribe the order in which these calls are made

# Binary Trees

## Traversing a Binary Tree

- There are different ways a binary tree may be traversed
- Definition template for a function on a (`btof X`):
  - a call to a function on a `X`
  - a call to process the left subtree
  - a call to process the right subtree
- The template does not prescribe the order in which these calls are made
- Three of the basic strategies:
  - `preorder traversal` First processes the value at the root then processes the left subtree and ends by processing the right subtree
  - `inorder traversal` First processes the left subtree then processes the value at the root and ends by processing the right subtree
  - `postorder traversal` First processes the left subtree then processes the right subtree and ends by processing the value at the root

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## Traversing a Binary Tree

- Consider extracting all the strings from a (bt of string)

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## Traversing a Binary Tree

- Consider extracting all the strings from a (btotf string)
- The returned value must also be data of arbitrary size: (listof string)

# Binary Trees

## Traversing a Binary Tree

- Consider extracting all the strings from a (btotf string)
- The returned value must also be data of arbitrary size: (listof string)
- Reason in terms of the structure of a binary tree
- If the tree is empty the returned value is the empty list

# Binary Trees

## Traversing a Binary Tree

- Consider extracting all the strings from a (bt of string)
- The returned value must also be data of arbitrary size: (list of string)
- Reason in terms of the structure of a binary tree
- If the tree is empty the returned value is the empty list
- Otherwise, we must decide how to traverse the tree
- The problem statement does not specify the order in which the strings must be returned
- Arbitrarily choose to return the list of strings following a preorder traversal

# Binary Trees

## Traversing a Binary Tree

- ```
(define E-BTSTRING '())  ;; Sample instances of (bt of X)
(define BTSTRING2
  (list "Pokemon" (list "Attack on Titan" (list "Dragon Ball" '() '())
                           (list "One Piece" '() '()))
        (list "Naruto" (list "Death Note" '() '()) (list "Detective Conan" '() '()))))
```

Binary Trees

Traversing a Binary Tree

- (define E-BTSTRING '()) ; Sample instances of (bt of X)
(define BTSTRING2
 (list "Pokemon" (list "Attack on Titan" (list "Dragon Ball" '() '())
 (list "One Piece" '() '())
 (list "Naruto" (list "Death Note" '() '()) (list "Detective Conan" '() '()))))
 - (define EBTSTRING-VAL '()) ; Sample expressions for max-bttnatnum
(define BTSTRING2-VAL
 (cons (first BTSTRING2) (append (btstring-extract (second BTSTRING2))
 (btstring-extract (third BTSTRING2))))))

Binary Trees

Traversing a Binary Tree

- (define E-BTSTRING '()) ; Sample instances of (btot X)
(define BTSTRING2
 (list "Pokemon" (list "Attack on Titan" (list "Dragon Ball" '() '())
 (list "One Piece" '() '()))
 (list "Naruto" (list "Death Note" '() '()) (list "Detective Conan" '() '()))))
 - ; (btot string) → (listof string) Purpose: Preorder of given (btot string)
(define (btstring-extract a-btstring)
 - (define EBTSTRING-VAL '()) ; Sample expressions for max-btnatnum
(define BTSTRING2-VAL
 (cons (first BTSTRING2) (append (btstring-extract (second BTSTRING2))
 (btstring-extract (third BTSTRING2))))))

Binary Trees

Traversing a Binary Tree

- (define E-BTSTRING '()) ; Sample instances of (btft X)
(define BTSTRING2
 (list "Pokemon" (list "Attack on Titan" (list "Dragon Ball" '() '())
 (list "One Piece" '() '()))
 (list "Naruto" (list "Death Note" '() '()) (list "Detective Conan" '() '()))))
 - ; (btft string) → (listof string) Purpose: Preorder of given (btft string)
(define (btstring-extract a-btstring)

Binary Trees

Traversing a Binary Tree

- (define E-BTSTRING '()) ; Sample instances of (btft X)
(define BTSTRING2
 (list "Pokemon" (list "Attack on Titan" (list "Dragon Ball" '() '())
 (list "One Piece" '() '()))
 (list "Naruto" (list "Death Note" '() '()) (list "Detective Conan" '() '()))))
- ;; (btft string) → (listof string) Purpose: Preorder of given (btft string)
(define (btstring-extract a-btstring)
- (if (empty? a-btstring)
 '()
 (cons (first a-btstring) (append (btstring-extract (second a-btstring))
 (btstring-extract (third a-btstring))))))
- (define EBTSTRING-VAL '()) ; Sample expressions for max-btnatnum
(define BTSTRING2-VAL
 (cons (first BTSTRING2) (append (btstring-extract (second BTSTRING2))
 (btstring-extract (third BTSTRING2))))))
- (check-expect (btstring-extract E-BTSTRING) EBTSTRING-VAL) ; Sample computations tests
(check-expect (btstring-extract BTSTRING2) BTSTRING2-VAL)
(check-expect ; Tests using sample values
 (btstring-extract
 (list "Attack on Titan" '() (list "Sailor Moon"
 (list "Fullmetal Alchemist" '() '())
 '()))
 (list "Attack on Titan" "Sailor Moon" "Fullmetal Alchemist")))
(check-expect
 (btstring-extract
 (list "This"
 (list "BT"
 (list "is" (list "like" (list "a" (list "list." '() '()) '()) '())
 '())
 '())
 (list "This" "BT" "is" "like" "a" "list.")))

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

The Maximum of a (bt of int)

- Consider finding the maximum in a (bt of int)

Binary Trees

The Maximum of a (bt of int)

- Consider finding the maximum in a (bt of int)
- What does the root value need to be compared to determine the maximum?

Binary Trees

The Maximum of a (bt of int)

- Consider finding the maximum in a (bt of int)
- What does the root value need to be compared to determine the maximum?
- The root value needs to be compared to the maximum of the left subtree and the maximum of the right subtree
- That means that a postorder traversal is a natural fit to this problem

Binary Trees

The Maximum of a (btot int)

- Consider finding the maximum in a (btot int)
- What does the root value need to be compared to determine the maximum?
- The root value needs to be compared to the maximum of the left subtree and the maximum of the right subtree
- That means that a postorder traversal is a natural fit to this problem
- Reason in terms of the structure of a (btot int)
- What is the answer if the (btot int) is empty?

Binary Trees

The Maximum of a (btot int)

- Consider finding the maximum in a (btot int)
- What does the root value need to be compared to determine the maximum?
- The root value needs to be compared to the maximum of the left subtree and the maximum of the right subtree
- That means that a postorder traversal is a natural fit to this problem
- Reason in terms of the structure of a (btot int)
- What is the answer if the (btot int) is empty?
- If the given (btot int) is empty then it contains no maximum number
- This informs us that we either need to define a subtype for a non-empty (btot int) or the function must throw an error if it is given a empty (btot int) as input
- On this occasion, we design using the second option

Binary Trees

The Maximum of a (btotf int)

- Consider finding the maximum in a (btotf int)
- What does the root value need to be compared to determine the maximum?
- The root value needs to be compared to the maximum of the left subtree and the maximum of the right subtree
- That means that a postorder traversal is a natural fit to this problem
- Reason in terms of the structure of a (btotf int)
- What is the answer if the (btotf int) is empty?
- If the given (btotf int) is empty then it contains no maximum number
- This informs us that we either need to define a subtype for a non-empty (btotf int) or the function must throw an error if it is given a empty (btotf int) as input
- On this occasion, we design using the second option
- If the given (btotf int) is not empty then the function needs to determine the maximum among the left subtree maximum, the right subtree maximum, and the root value
- Care must be taken because any of the subtrees or both may be empty
- A recursive call must not be made with an empty subtree to avoid having the function throw an error

Binary Trees

The Maximum of a (bt of int)

- There are 5 conditions:

(bt of int)	Return
empty	throw error
leaf	root value
left subtree empty	max of the root value and the right subtree max
right subtree empty	max of the root value and the left subtree max
neither subtree empty	max of root value, left subtree max, and right subtree max

Binary Trees

The Maximum of a (bt of int)

- There are 5 conditions:

(bt of int)	Return
empty	throw error
leaf	root value
left subtree empty	max of the root value and the right subtree max
right subtree empty	max of the root value and the left subtree max
neither subtree empty	max of root value, left subtree max, and right subtree max

- Define 5 sample (bt of int) instances as follows:

```
;; Sample instances of (bt of int)
(define E-BTINT '()) ; empty tree
(define BTINT1  (list 200 '() '())) ; leaf
(define BTINT2  (list 100   ; neither subtree is empty
                      (list 1
                            (list 3 '() '())
                            (list 4 '() '())))
                      (list 2
                            (list 5 '() '())
                            (list 6 '() '())))
                      )
(define BTINT3  (list -10   ; left subtree is empty
                      '()
                      (list 78 '() '())))
(define BTINT4  (list -8    ; right subtree is empty
                      (list -5
                            (list 47 '() '())
                            (list -1 '() '())))
                      '()))
```

Binary Trees

The Maximum of a (bt of int)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ; Sample expressions for max-btint

```
(define BTINT1-VAL  (first BTINT1))
(define BTINT2-VAL  (max (first BTINT2) (max-btint (second BTINT2))
                           (max-btint (third BTINT2))))
(define BTINT3-VAL  (max (first BTINT3) (max-btint (third BTINT3))))
(define BTINT4-VAL  (max (first BTINT4) (max-btint (second BTINT4))))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

The Maximum of a (bt of int)

- `;; (bt of int) → int throws error Purpose: Find (bt of int) maximum
(define (max-btint a-btint))`

- `;; Sample expressions for max-btint
(define BTINT1-VAL (first BTINT1))
(define BTINT2-VAL (max (first BTINT2) (max-btint (second BTINT2))
 (max-btint (third BTINT2))))
(define BTINT3-VAL (max (first BTINT3) (max-btint (third BTINT3))))
(define BTINT4-VAL (max (first BTINT4) (max-btint (second BTINT4))))`

Binary Trees

The Maximum of a (bt of int)

- `; ; (bt of int) → int throws error` Purpose: Find (bt of int) maximum
`(define (max-btint a-btint)`

- `; Sample expressions for max-btint`
`(define BTINT1-VAL (first BTINT1))`
`(define BTINT2-VAL (max (first BTINT2) (max-btint (second BTINT2)))`
`(max-btint (third BTINT2))))`
`(define BTINT3-VAL (max (first BTINT3) (max-btint (third BTINT3))))`
`(define BTINT4-VAL (max (first BTINT4) (max-btint (second BTINT4))))`
- `; Tests using sample computations for max-btint`
`(check-expect (max-btint BTINT1) BTINT1-VAL)`
`(check-expect (max-btint BTINT2) BTINT2-VAL)`
`(check-expect (max-btint BTINT3) BTINT3-VAL)`
`(check-expect (max-btint BTINT4) BTINT4-VAL)`
`; Tests using sample values for max-btint`
`(check-error`
`(max-btint E-BTINT)`
`"An empty (bt of natnum) has no maximum value."`
`(check-expect (max-btint (list -67`
`(list -50 '() '())`
`(list -8 '() '()))))`

Binary Trees

The Maximum of a (bt of int)

- `;; (bt of int) → int` throws error Purpose: Find (bt of int) maximum
`(define (max-btint a-btint)`
 `(cond [(empty? a-btint) (error "An empty (bt of int) has no maximum value.")]`
 `[(and (empty? (second a-btint)) (empty? (third a-btint))) (first a-btint)]`
 `[(empty? (second a-btint)) (max (first a-btint) (max-btint (third a-btint)))]`
 `[(empty? (third a-btint)) (max (first a-btint) (max-btint (second a-btint)))]`
 `[else (max (first a-btint)`
 `(max-btint (second a-btint))`
 `(max-btint (third a-btint)))]]))`
- `;; Sample expressions for max-btint`
`(define BTINT1-VAL (first BTINT1))`
`(define BTINT2-VAL (max (first BTINT2) (max-btint (second BTINT2))`
 `(max-btint (third BTINT2))))`
`(define BTINT3-VAL (max (first BTINT3) (max-btint (third BTINT3))))`
`(define BTINT4-VAL (max (first BTINT4) (max-btint (second BTINT4))))`
- `;; Tests using sample computations for max-btint`
`(check-expect (max-btint BTINT1) BTINT1-VAL)`
`(check-expect (max-btint BTINT2) BTINT2-VAL)`
`(check-expect (max-btint BTINT3) BTINT3-VAL)`
`(check-expect (max-btint BTINT4) BTINT4-VAL)`
`;; Tests using sample values for max-btint`
`(check-error`
 `(max-btint E-BTINT)`
 `"An empty (bt of natnum) has no maximum value.")`
`(check-expect (max-btint (list -67`
 `(list -50 '() '())`
 `(list -8 '() '()))))`

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

Homework

- Problems: 181–184

Binary Trees

Binary Search Trees

- A database stores organized information in a computer program
- Consider representing a database of criminals:

```
;; A criminal record is a structure, (make-cr natnum string string),  
;; with an id number, a nickname, and a name.  
(define-struct cr (id nickname name))
```

- Sample cr instances:

```
(define TEFLONDON (make-cr 241  
                         "The Teflon Don"  
                         "John Gotti Jr."))  
  
(define BABYFACE (make-cr 77  
                           "Babyface Nelson"  
                           "Lester Joseph Gillis"))  
  
(define SCARFACE (make-cr 23  
                           "Scarface"  
                           "Alphonse Gabriel Capone"))  
  
(define DILLINGER (make-cr 675  
                           "Gentleman John"  
                           "John Herbert Dillinger"))  
  
(define BUGSY (make-cr 874 "Bugsy" "Benjamin Siegel"))  
(define PAULIE (make-cr 1  
                           "Big Pauli"  
                           "Paul Castellano"))  
  
(define VITO (make-cr 55 "Don Vitone" "Vito Genovese"))
```



Binary Trees

A (listof cr) Representation

- We must think about is how will the database be represented
- Given that the number of criminals is not known in advance a database is of arbitrary size
- This means we need a recursive data structure to represent it

Binary Trees

A (listof cr) Representation

- We must think about is how will the database be represented
- Given that the number of criminals is not known in advance a database is of arbitrary size

- This means we need a recursive data structure to represent it
- We may define a criminal database and sample instances as follows:

```
;; A criminal database is a (listof cr).
```

```
;; Sample instances
```

```
(define CR-DB0 '())
```

```
(define CR-DB1 (list TEFLONDON BABYFACE SCARFACE DILLINGER BUGSY))
```

- Is this a good and efficient representation?

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (listof cr) Representation

- Consider the problem of returning a cr in a database given an id number

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (listof cr) Representation

- Consider the problem of returning a cr in a database given an id number
- Think in terms of the structure of the database
- What is the answer if the database is empty?

Binary Trees

A (listof cr) Representation

- Consider the problem of returning a cr in a database given an id number
- Think in terms of the structure of the database
- What is the answer if the database is empty?
- In this case, the wanted cr does not exist
- We define the following cr to signal that the a cr with the given id number does not exists:

```
(define DNE-CR (make-cr 0 "DNE" "CR DOES NOT EXIST"))
```

- When the database is empty the answer is DNE-CR
- What is the answer if the database is not empty?

Binary Trees

A (listof cr) Representation

- Consider the problem of returning a cr in a database given an id number
- Think in terms of the structure of the database
- What is the answer if the database is empty?
- In this case, the wanted cr does not exist
- We define the following cr to signal that the a cr with the given id number does not exists:

```
(define DNE-CR (make-cr 0 "DNE" "CR DOES NOT EXIST"))
```

- When the database is empty the answer is DNE-CR
- What is the answer if the database is not empty?
- The id of the first cr is compared with the given id
- If they match, then the first cr is returned
- Otherwise, the rest of the database is recursively searched

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (listof cr) Representation

- ```
;; Sample expressions for get-record-locr
(define GR1-VAL DNE-CR)
(define GR2-VAL TEF-LONDON)
(define GR3-VAL (get-record-locr 675 (rest CR-DB1)))
```

# Binary Trees

## A (listof cr) Representation

- ```
;; number (listof cr) → cr
;; Purpose: Return the cr with the given id Typo in textbook
(define (get-record-locr id a-locr)
```
- ```
;; Sample expressions for get-record-locr
(define GR1-VAL DNE-CR)
(define GR2-VAL TEF-LONDON)
(define GR3-VAL (get-record-locr 675 (rest CR-DB1)))
```

# Binary Trees

## A (listof cr) Representation

- ```
;; number (listof cr) → cr
;; Purpose: Return the cr with the given id Typo in textbook
(define (get-record-locr id a-loctr)
```
- ```
;; Sample expressions for get-record-loctr
(define GR1-VAL DNE-CR)
(define GR2-VAL TEFLONDON)
(define GR3-VAL (get-record-loctr 675 (rest CR-DB1)))
```
- ```
;; Tests using sample computations for get-record-loctr
(check-expect (get-record-loctr 899 CR-DB1) GR1-VAL)
(check-expect (get-record-loctr 241 CR-DB1) GR2-VAL)
(check-expect (get-record-loctr 675 CR-DB1) GR3-VAL)
```
- ```
;; Tests using sample values for get-record-loctr
(check-expect (get-record-loctr 55 (list PAULIE VITO)) VITO)
(check-expect (get-record-loctr 1 (list PAULIE VITO)) PAULIE)
```

# Binary Trees

## A (listof cr) Representation

- ```
;; number (listof cr) → cr
;; Purpose: Return the cr with the given id Typo in textbook
(define (get-record-locr id a-loctr)
```
- ```
(cond [(empty? a-loctr) DNE-CR]
 [(= (cr-id (first a-loctr)) id) (first a-loctr)]
 [else (get-record-locr id (rest a-loctr))]))
```
- ```
;; Sample expressions for get-record-locr
(define GR1-VAL DNE-CR)
(define GR2-VAL TEFLONDON)
(define GR3-VAL (get-record-locr 675 (rest CR-DB1)))
```
- ```
;; Tests using sample computations for get-record-locr
(check-expect (get-record-locr 899 CR-DB1) GR1-VAL)
(check-expect (get-record-locr 241 CR-DB1) GR2-VAL)
(check-expect (get-record-locr 675 CR-DB1) GR3-VAL)
```
- ```
;; Tests using sample values for get-record-locr
(check-expect (get-record-locr 55 (list PAULIE VITO)) VITO)
(check-expect (get-record-locr 1 (list PAULIE VITO)) PAULIE)
```

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees
- We may define the same database from the previous subsection as:

```
(define CR-DB3 (list DILLINGER
                        (list BABYFACE '() (list SCARFACE '() '()))
                        (list TEFLONDON (list BUGSY '() '()) '())))
```

- DILLINGER is at the root of the tree and the rest of the criminal records are evenly distributed among the subtrees
- Consider how to retrieve a criminal record given an id number
- Reason in terms of the structure of a binary tree

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees
- We may define the same database from the previous subsection as:

```
(define CR-DB3 (list DILLINGER
                      (list BABYFACE '() (list SCARFACE '() '()))
                      (list TEFLONDON (list BUGSY '() '()) '())))
```

- DILLINGER is at the root of the tree and the rest of the criminal records are evenly distributed among the subtrees
- Consider how to retrieve a criminal record given an id number
- Reason in terms of the structure of a binary tree
- If the given binary tree is empty then the answer is DNE-CR
- What is the answer if the binary tree is not empty?

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees
- We may define the same database from the previous subsection as:

```
(define CR-DB3 (list DILLINGER
                      (list BABYFACE '() (list SCARFACE '() '()))
                      (list TEFLONDON (list BUGSY '() '()) '())))
```

- DILLINGER is at the root of the tree and the rest of the criminal records are evenly distributed among the subtrees
- Consider how to retrieve a criminal record given an id number
- Reason in terms of the structure of a binary tree
- If the given binary tree is empty then the answer is DNE-CR
- What is the answer if the binary tree is not empty?
- The tree must be traversed starting with the root's cr
- A preorder traversal is appropriate

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees
- We may define the same database from the previous subsection as:

```
(define CR-DB3 (list DILLINGER
                      (list BABYFACE '() (list SCARFACE '() '()))
                      (list TEFLONDON (list BUGSY '() '()) '())))
```

- DILLINGER is at the root of the tree and the rest of the criminal records are evenly distributed among the subtrees
- Consider how to retrieve a criminal record given an id number
- Reason in terms of the structure of a binary tree
- If the given binary tree is empty then the answer is DNE-CR
- What is the answer if the binary tree is not empty?
- The tree must be traversed starting with the root's cr
- A preorder traversal is appropriate
- If the root's cr's id matches the given id then the answer is the root cr

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees
- We may define the same database from the previous subsection as:

```
(define CR-DB3 (list DILLINGER
                      (list BABYFACE '() (list SCARFACE '() '()))
                      (list TEFLONDON (list BUGSY '() '()) '())))
```

- DILLINGER is at the root of the tree and the rest of the criminal records are evenly distributed among the subtrees
- Consider how to retrieve a criminal record given an id number
- Reason in terms of the structure of a binary tree
- If the given binary tree is empty then the answer is DNE-CR
- What is the answer if the binary tree is not empty?
- The tree must be traversed starting with the root's cr
- A preorder traversal is appropriate
- If the root's cr's id matches the given id then the answer is the root cr
- If they do not match then we traverse the left subtree first
- If the result is DNE-CR then the right subtree must be traversed

Binary Trees

A (bt of cr) Representation

- Another representation: (bt of cr)
- The idea here is to have a cr at the root and distribute the remaining crs among the two subtrees
- We may define the same database from the previous subsection as:

```
(define CR-DB3 (list DILLINGER
                        (list BABYFACE '() (list SCARFACE '() '()))
                        (list TEFLONDON (list BUGSY '() '() '()))))
```

- DILLINGER is at the root of the tree and the rest of the criminal records are evenly distributed among the subtrees
- Consider how to retrieve a criminal record given an id number
- Reason in terms of the structure of a binary tree
- If the given binary tree is empty then the answer is DNE-CR
- What is the answer if the binary tree is not empty?
- The tree must be traversed starting with the root's cr
- A preorder traversal is appropriate
- If the root's cr's id matches the given id then the answer is the root cr
- If they do not match then we traverse the left subtree first
- If the result is DNE-CR then the right subtree must be traversed
- Otherwise, the answer is the result obtained from traversing the left subtree

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (btocr cr) Representation

- ; Sample expressions for get-record-btocr
 - (define GRBT1-VAL DNE-CR)
 - (define GRBT2-VAL DILLINGER)
 - (define GRBT3-VAL (get-record-btocr 874 (third CR-DB3)))
 - (define GRBT4-VAL (get-record-btocr 77 (second CR-DB3)))

Binary Trees

A (btot cr) Representation

- ```
;; number (btot cr) → cr
;; Purpose: To return list of the even numbers in the given list
(define (get-record-btotr id a-btotr)
```

- ```
;; Sample expressions for get-record-btotr
(define GRBT1-VAL DNE-CR)
(define GRBT2-VAL DILLINGER)
(define GRBT3-VAL (get-record-btotr 874 (third CR-DB3)))
(define GRBT4-VAL (get-record-btotr 77 (second CR-DB3)))
```

Binary Trees

A (btocr cr) Representation

- ; number (btocr cr) → cr
; Purpose: To return list of the even numbers in the given list

```
(define (get-record-btocr id a-btocr)
```
- ; Sample expressions for get-record-btocr

```
(define GRBT1-VAL DNE-CR)
(define GRBT2-VAL DILLINGER)
(define GRBT3-VAL (get-record-btocr 874 (third CR-DB3)))
(define GRBT4-VAL (get-record-btocr 77 (second CR-DB3)))
```
- ; Tests using sample computations for get-record-btocr

```
(check-expect (get-record-btocr 899 CR-DB0) GRBT1-VAL)
(check-expect (get-record-btocr 675 CR-DB3) GRBT2-VAL)
(check-expect (get-record-btocr 874 CR-DB3) GRBT3-VAL)
(check-expect (get-record-btocr 77 CR-DB3) GRBT4-VAL)
; Tests using sample values for get-record-locr
(check-expect (get-record-btocr 55 (list PAULIE '() (list VITO '() '())))
              VITO)
(check-expect (get-record-btocr 1 (list PAULIE '() (list VITO '() '())))
              PAULIE)
(check-expect (get-record-btocr 16 (list BABYFACE
                                         (list SCARFACE '() '())
                                         (list BUGSY '() '())))
              DNE-CR)
```

Binary Trees

A (btot cr) Representation

- ; number (btot cr) → cr
;; Purpose: To return list of the even numbers in the given list

```
(define (get-record-btocr id a-btocr)
```
- (cond [(empty? a-btocr) DNE-CR]
 [(= (cr-id (first a-btocr)) id) (first a-btocr)]
 [(equal? (get-record-btocr id (second a-btocr)) DNE-CR)
 (get-record-btocr id (third a-btocr))] ← repeated work
 [else (get-record-btocr id (second a-btocr))]))
- ; Sample expressions for get-record-btocr

```
(define GRBT1-VAL DNE-CR)
(define GRBT2-VAL DILLINGER)
(define GRBT3-VAL (get-record-btocr 874 (third CR-DB3)))
(define GRBT4-VAL (get-record-btocr 77 (second CR-DB3)))
```
- ; Tests using sample computations for get-record-btocr

```
(check-expect (get-record-btocr 899 CR-DB0) GRBT1-VAL)
(check-expect (get-record-btocr 675 CR-DB3) GRBT2-VAL)
(check-expect (get-record-btocr 874 CR-DB3) GRBT3-VAL)
(check-expect (get-record-btocr 77 CR-DB3) GRBT4-VAL)
; Tests using sample values for get-record-locr
(check-expect (get-record-btocr 55 (list PAULIE '() (list VITO '() '())))
              VITO)
(check-expect (get-record-btocr 1 (list PAULIE '() (list VITO '() '())))
              PAULIE)
(check-expect (get-record-btocr 16 (list BABYFACE
                                         (list SCARFACE '() '())
                                         (list BUGSY '() '())))
              DNE-CR)
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (bstof cr) Representation

- Can searching the database be made faster?

Binary Trees

A (bstof cr) Representation

- Can searching the database be made faster?
- We can avoid traversing the whole tree searching for a cr if all the ids less than the root are in the left subtree and all the ids greater than the root are in the right subtree
- In this manner if the given id does not match the root then only the left or only the right subtree needs to be traversed.

Binary Trees

A (bstof cr) Representation

- Can searching the database be made faster?
- We can avoid traversing the whole tree searching for a cr if all the ids less than the root are in the left subtree and all the ids greater than the root are in the right subtree
- In this manner if the given id does not match the root then only the left or only the right subtree needs to be traversed.
- A (bstof X) that satisfies these *invariant* properties is called a *binary search tree*:

```
;; A (bstof X) is either
;; 1. '()
;; 2. (list X (bstof X) (btsof X))
;;   SUCH THAT
;;     A. All Xs in the left subtree are less than the root X
;;     B. All Xs in the right subtree are greater than the root X
```

- Observe that the data definition lists the invariant properties. In this manner, it is clear to all readers and problem solvers what a (bstof X)
- (bstof X) is a subtype of (btsof X)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

- How do we search for the needed cr?

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

- How do we search for the needed cr?
- Reason about the structure of a (bstof X)

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

- How do we search for the needed cr?
- Reason about the structure of a (bstof X)
- If the (bstof X) is empty the answer is DNE-CR

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

- How do we search for the needed cr?
- Reason about the structure of a (bstof X)
- If the (bstof X) is empty the answer is DNE-CR
- If the given id matches the id at the root then the answer is the root cr

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

- How do we search for the needed cr?
- Reason about the structure of a (bstof X)
- If the (bstof X) is empty the answer is DNE-CR
- If the given id matches the id at the root then the answer is the root cr
- If the given id is less than the root's id then the left subtree is recursively searched

Binary Trees

A (bstof cr) Representation

- The sample database:

```
(define CR-DB4 (list TEFLONDON
                        (list BABYFACE
                                (list SCARFACE '() '())
                                '())
                            (list DILLINGER
                                '()
                                (list BUGSY '() '()))))
```

- How do we search for the needed cr?
- Reason about the structure of a (bstof X)
- If the (bstof X) is empty the answer is DNE-CR
- If the given id matches the id at the root then the answer is the root cr
- If the given id is less than the root's id then the left subtree is recursively searched
- Otherwise, the right subtree is recursively searched

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

A (bstof cr) Representation

- ;; Sample expressions for get-record-bstocr
 - (define GRBST1-VAL DNE-CR)
 - (define GRBST2-VAL (first CR-DB4))
 - (define GRBST3-VAL (get-record-bstocr 23 (second CR-DB4)))
 - (define GRBST4-VAL (get-record-bstocr 675 (third CR-DB4)))

Binary Trees

A (bstof cr) Representation

- ```
;; number (bstof cr) → cr
;; Purpose: To return the cr with the given id if it
;; exists in the given (bstof cr)
(define (get-record-bstocr id a-bstocr)
```

- ```
;; Sample expressions for get-record-bstocr
(define GRBST1-VAL DNE-CR)
(define GRBST2-VAL (first CR-DB4))
(define GRBST3-VAL (get-record-bstocr 23 (second CR-DB4)))
(define GRBST4-VAL (get-record-bstocr 675 (third CR-DB4)))
```

Binary Trees

A (bstof cr) Representation

- ```
;; number (bstof cr) → cr
;; Purpose: To return the cr with the given id if it
;; exists in the given (bstof cr)
(define (get-record-bstocr id a-bstocr)
```
- ```
;; Sample expressions for get-record-bstocr
(define GRBST1-VAL DNE-CR)
(define GRBST2-VAL (first CR-DB4))
(define GRBST3-VAL (get-record-bstocr 23 (second CR-DB4)))
(define GRBST4-VAL (get-record-bstocr 675 (third CR-DB4)))

;; Tests using sample computations for get-record-bstocr
(check-expect (get-record-bstocr 899 CR-DB0) GRBST1-VAL)
(check-expect (get-record-bstocr 241 CR-DB4) GRBST2-VAL)
(check-expect (get-record-bstocr 23 CR-DB4) GRBST3-VAL)
(check-expect (get-record-bstocr 675 CR-DB4) GRBST4-VAL)

;; Tests using sample values for get-record-bstocr
(check-expect (get-record-bstocr 55
                                (list PAULIE '() (list VITO '() '())))
              VITO)
(check-expect (get-record-bstocr 1
                                (list PAULIE '() (list VITO '() '())))
              PAULIE)
(check-expect (get-record-bstocr 14
                                (list PAULIE '() (list VITO '() '())))
              DNE-CR)
```

Binary Trees

A (bstof cr) Representation

- ```
;; number (bstof cr) → cr
;; Purpose: To return the cr with the given id if it
;; exists in the given (bstof cr)
(define (get-record-bstocr id a-bstocr)
```
- ```
(cond [(empty? a-bstocr) DNE-CR]
        [ (= (cr-id (first a-bstocr)) id) (first a-bstocr)]
        [ (< id (cr-id (first a-bstocr)))
          (get-record-bstocr id (second a-bstocr))]
        [else (get-record-bstocr id (third a-bstocr))]))
```
- ```
; Sample expressions for get-record-bstocr
(define GRBST1-VAL DNE-CR)
(define GRBST2-VAL (first CR-DB4))
(define GRBST3-VAL (get-record-bstocr 23 (second CR-DB4)))
(define GRBST4-VAL (get-record-bstocr 675 (third CR-DB4)))
```
- ```
; Tests using sample computations for get-record-bstocr
(check-expect (get-record-bstocr 899 CR-DB0) GRBST1-VAL)
(check-expect (get-record-bstocr 241 CR-DB4) GRBST2-VAL)
(check-expect (get-record-bstocr 23 CR-DB4) GRBST3-VAL)
(check-expect (get-record-bstocr 675 CR-DB4) GRBST4-VAL)
; Tests using sample values for get-record-bstocr
(check-expect (get-record-bstocr 55
                                (list PAULIE '() (list VITO '() '())))
              VITO)
(check-expect (get-record-bstocr 1
                                (list PAULIE '() (list VITO '() '())))
              PAULIE)
(check-expect (get-record-bstocr 14
                                (list PAULIE '() (list VITO '() '())))
              DNE-CR)
```

Binary Trees

Abstract Running Time

- We have three different ways to search a criminal database based on three different representations
- Which one should we choose?

Binary Trees

Abstract Running Time

- We have three different ways to search a criminal database based on three different representations
- Which one should we choose?
- The answer depends on the criteria that must be met

Binary Trees

Abstract Running Time

- We have three different ways to search a criminal database based on three different representations
- Which one should we choose?
- The answer depends on the criteria that must be met
- If the most important criteria is code simplicity then it is likely that the representation using a (listof cr) is the best
- Most of the time, however, it is more important for a program to be efficient in terms of execution time, memory usage, or both

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

Abstract Running Time

- Let us focus on execution time
- How can we compare two programs to decide which one is more efficient in terms of execution time?

Binary Trees

Abstract Running Time

- Let us focus on execution time
- How can we compare two programs to decide which one is more efficient in terms of execution time?
- One way to measure efficiency is by timing the programs
- In ISL+ we can time programs using the time function
- The time function takes as input an expression to evaluate and returns its value
- Before returning the value of the expression it prints information about execution time in milliseconds including CPU time

Binary Trees

Abstract Running Time

- Let us focus on execution time
- How can we compare two programs to decide which one is more efficient in terms of execution time?
- One way to measure efficiency is by timing the programs
- In ISL+ we can time programs using the time function
- The time function takes as input an expression to evaluate and returns its value
- Before returning the value of the expression it prints information about execution time in milliseconds including CPU time
- (`define LST1 (build-random-list 5000)`)

```
(define SORTED1 (time (sort-lon LST1)))  
(define SORTED2 (time (sort      LST1 <)))
```

Binary Trees

Abstract Running Time

- Let us focus on execution time
- How can we compare two programs to decide which one is more efficient in terms of execution time?
- One way to measure efficiency is by timing the programs
- In ISL+ we can time programs using the time function
- The time function takes as input an expression to evaluate and returns its value
- Before returning the value of the expression it prints information about execution time in milliseconds including CPU time
- (`define LST1 (build-random-list 5000)`)

```
(define SORTED1 (time (sort-lon LST1)))  
(define SORTED2 (time (sort      LST1 <)))
```

- Running the program 5 times:

Run	in sort	sort
1	26406	15
2	25922	0
3	25719	16
4	25109	16
5	26344	47

- ISL+'s sort function is faster than our sort-lon function
- Execution timing is an unreliable measure

Binary Trees

Abstract Running Time

- There is a better way to compare the expected execution time of programs
- Count the number of operations performed in relation to input size
- For example, count recursive) calls or the number of comparisons
- These do not vary among different runs nor among different computer

Binary Trees

Abstract Running Time

- There is a better way to compare the expected execution time of programs
- Count the number of operations performed in relation to input size
- For example, count recursive) calls or the number of comparisons
- These do not vary among different runs nor among different computer
- Fewer operations \Rightarrow faster regardless of the computer used
- This is called abstract running time or the complexity

Binary Trees

Abstract Running Time

- There is a better way to compare the expected execution time of programs
- Count the number of operations performed in relation to input size
- For example, count recursive) calls or the number of comparisons
- These do not vary among different runs nor among different computer
- Fewer operations \Rightarrow faster regardless of the computer used
- This is called abstract running time or the complexity
- We do not count every single operation done by the computer
- We count abstract operations that may involve many computer operations

Binary Trees

Abstract Running Time

- There is a better way to compare the expected execution time of programs
- Count the number of operations performed in relation to input size
- For example, count recursive) calls or the number of comparisons
- These do not vary among different runs nor among different computer
- Fewer operations \Rightarrow faster regardless of the computer used
- This is called abstract running time or the complexity
- We do not count every single operation done by the computer
- We count abstract operations that may involve many computer operations
- How abstract operations are implemented varies from one computer to another
- The number of computer operations is proportional to some constant k
- This constant of proportionality is different for different computers

Binary Trees

Abstract Running Time

- There is a better way to compare the expected execution time of programs
- Count the number of operations performed in relation to input size
- For example, count recursive) calls or the number of comparisons
- These do not vary among different runs nor among different computer
- Fewer operations \Rightarrow faster regardless of the computer used
- This is called abstract running time or the complexity
- We do not count every single operation done by the computer
- We count abstract operations that may involve many computer operations
- How abstract operations are implemented varies from one computer to another
- The number of computer operations is proportional to some constant k
- This constant of proportionality is different for different computers
- Typically, we are concerned with the worst-case scenario
- Number of abstract operations is described by a function on the input size
- Input size is $n \Rightarrow$ the complexity is the highest power of n in this function
- Big O notation used to describe complexity in terms of the input size
- If $5n^2+2n+1$ abstract operations then complexity is $O(n^2)$

Binary Trees

Abstract Running Time

- There is a better way to compare the expected execution time of programs
- Count the number of operations performed in relation to input size
- For example, count recursive) calls or the number of comparisons
- These do not vary among different runs nor among different computer
- Fewer operations \Rightarrow faster regardless of the computer used
- This is called abstract running time or the complexity
- We do not count every single operation done by the computer
- We count abstract operations that may involve many computer operations
- How abstract operations are implemented varies from one computer to another
- The number of computer operations is proportional to some constant k
- This constant of proportionality is different for different computers
- Typically, we are concerned with the worst-case scenario
- Number of abstract operations is described by a function on the input size
- Input size is $n \Rightarrow$ the complexity is the highest power of n in this function
- Big O notation used to describe complexity in terms of the input size
- If $5n^2+2n+1$ abstract operations then complexity is $O(n^2)$
- ```
(define (build-random-list n)
 (if (= n 0)
 '()
 (cons (random 10000000) (build-random-list (sub1 n)))))
```
- `build-random-list` is  $O(n)$

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## The Complexity of Searching the Criminal Database

- To determine if any of the criminal-record database implementations is superior we compare their abstract running time

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## The Complexity of Searching the Criminal Database

- To determine if any of the criminal-record database implementations is superior we compare their abstract running time
- Consider the database represented as a (listof cr)

# Binary Trees

## The Complexity of Searching the Criminal Database

- To determine if any of the criminal-record database implementations is superior we compare their abstract running time
- Consider the database represented as a (listof cr)
- In the best case get-record-locr is called only once
- This occurs when the function is called with the empty list or when the record searched for is the first one in the list
- Therefore, the complexity is  $O(1)$ , or simply  $O(k)$

# Binary Trees

## The Complexity of Searching the Criminal Database

- To determine if any of the criminal-record database implementations is superior we compare their abstract running time
- Consider the database represented as a (listof cr)
- In the best case get-record-locr is called only once
- This occurs when the function is called with the empty list or when the record searched for is the first one in the list
- Therefore, the complexity is  $O(1)$ , or simply  $O(k)$
- What is the worse case scenario?

# Binary Trees

## The Complexity of Searching the Criminal Database

- To determine if any of the criminal-record database implementations is superior we compare their abstract running time
- Consider the database represented as a (listof cr)
- In the best case get-record-locr is called only once
- This occurs when the function is called with the empty list or when the record searched for is the first one in the list
- Therefore, the complexity is  $O(1)$ , or simply  $O(k)$
- What is the worse case scenario?
- Searching for an id number not in the database means that the function is called once for every criminal record
- If the given list has  $n$  records then the complexity is  $O(n)$
- This is a linear function
- If the size of the input is doubled then the number of operations performed is also doubled

# Binary Trees

## The Complexity of Searching the Criminal Database

- Let us now consider the complexity for using a (btot cr)

# Binary Trees

## The Complexity of Searching the Criminal Database

- Let us now consider the complexity for using a (btocr cr)
- In the best case get-record-btocr is called only once:  $O(k)$

# Binary Trees

## The Complexity of Searching the Criminal Database

- Let us now consider the complexity for using a (btocr cr)
- In the best case get-record-btocr is called only once:  $O(k)$
- The worst-case scenario is searching for an id number that is not in the tree

# Binary Trees

## The Complexity of Searching the Criminal Database

- Let us now consider the complexity for using a (btocr)
- In the best case get-record-btocr is called only once:  $O(k)$
- The worst-case scenario is searching for an id number that is not in the tree
- The left subtree is traversed to determine that the left subtree does not contain the given id number
- The right subtree is traversed
- The number of calls to get-record-btocr is equal to the number of nodes in the binary tree:  $O(n)$
- The same complexity as using a (listof cr)
- Using a binary tree provides no improvement

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

## Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## The Complexity of Searching the Criminal Database

- Complexity using a (bst of cr)?

# Binary Trees

## The Complexity of Searching the Criminal Database

- Complexity using a (bstof cr)?
- In the best case get-record-bstocr is only called once:  $O(k)$

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

## Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## The Complexity of Searching the Criminal Database

- Complexity using a (bst of cr)?
- In the best case `get-record-bstocr` is only called once:  $O(k)$
- Analyzing the worst-case scenario is more subtle

# Binary Trees

## The Complexity of Searching the Criminal Database

- Complexity using a (bstof cr)?
- In the best case get-record-bstocr is only called once:  $O(k)$
- Analyzing the worst-case scenario is more subtle
- It occurs when a search is performed for an id number that is larger (or smaller) than any id number in the binary search tree
- A careless glance at the code may suggest that at each step half of the criminal records are eliminated from the search because only the left or the right subtree is traversed
- Is it guaranteed that half of the criminal records are eliminated from the search every time the function is called?

# Binary Trees

## The Complexity of Searching the Criminal Database

- Complexity using a (bstof cr)?
- In the best case get-record-bstocr is only called once:  $O(k)$
- Analyzing the worst-case scenario is more subtle
- It occurs when a search is performed for an id number that is larger (or smaller) than any id number in the binary search tree
- A careless glance at the code may suggest that at each step half of the criminal records are eliminated from the search because only the left or the right subtree is traversed
- Is it guaranteed that half of the criminal records are eliminated from the search every time the function is called?
- Consider the following database:

```
(define BST-LST
 (list
 PAULIE
 '()
 (list SCARFACE
 '()
 (list VITO
 '()
 (list BABYFACE
 '()
 (list TEFLONDON
 '()
 (list DILLINGER
 '()
 (list BUGSY '() '()))))))))
```

- Observe that every left subtree is empty
- This means that the structure of this binary search tree is similar to the structure of a list:  $O(n)$
- All three of our database representations have a searching function with the same complexity
- None are expected to always be better than any of the others

# Binary Trees

## The Complexity of Searching the Criminal Database

- When using binary search trees the problem is that in the worst case only the record at the root is eliminated from the search
- Thus, forcing the examination of all records in the database

# Binary Trees

## The Complexity of Searching the Criminal Database

- When using binary search trees the problem is that in the worst case only the record at the root is eliminated from the search
- Thus, forcing the examination of all records in the database
- To overcome this problem we can represent the database using a *balanced* binary search tree
- A balanced binary tree is one in which the height of the left and right subtree of any node differ by no more than 1
- The descendants of a node are roughly evenly distributed in its subtrees

# Binary Trees

## The Complexity of Searching the Criminal Database

- When using binary search trees the problem is that in the worst case only the record at the root is eliminated from the search
- Thus, forcing the examination of all records in the database
- To overcome this problem we can represent the database using a *balanced* binary search tree
- A balanced binary tree is one in which the height of the left and right subtree of any node differ by no more than 1
- The descendants of a node are roughly evenly distributed in its subtrees
- A balanced binary search tree:

```
;; A (bbstof X) is either
;; 1. '()
;; 2. (list X (bbstof X) (bbstof X))
;; SUCH THAT
;; A. All Xs in the left subtree are less than the root X
;; B. All Xs in the right subtree are greater than the root X
;; C. (<= (- (height left-subtree) (height right-subtree)) 1)
```

- The third invariant guarantees that the nodes are evenly distributed a
- At each step of the search roughly half of the nodes left to explore are eliminated

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

## Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## Creating a Balanced Binary Search Tree

- Can a balanced binary search tree be created in the first place?

# Binary Trees

## Creating a Balanced Binary Search Tree

- Can a balanced binary search tree be created in the first place?
- Assume that we have a (`listof cr`) sorted in nondecreasing order by id number
- The middle element of the list must be the root of the balanced binary search tree

# Binary Trees

## Creating a Balanced Binary Search Tree

- Can a balanced binary search tree be created in the first place?
- Assume that we have a (`listof cr`) sorted in nondecreasing order by id number
- The middle element of the list must be the root of the balanced binary search tree
- All the `crs` before the middle element must be placed in the left (balanced) subtree
- All the `crs` after the middle element must be placed in the right (balanced) subtree

# Binary Trees

## Creating a Balanced Binary Search Tree

- Can a balanced binary search tree be created in the first place?
- Assume that we have a (`listof cr`) sorted in nondecreasing order by id number
- The middle element of the list must be the root of the balanced binary search tree
- All the `crs` before the middle element must be placed in the left (balanced) subtree
- All the `crs` after the middle element must be placed in the right (balanced) subtree
- The middle element is indexed by (`quotient (length a-locr) 2`)
- How then are the elements before and after the middle element referenced?

# Binary Trees

## Creating a Balanced Binary Search Tree

- Can a balanced binary search tree be created in the first place?
- Assume that we have a (listof cr) sorted in nondecreasing order by id number
- The middle element of the list must be the root of the balanced binary search tree
- All the crs before the middle element must be placed in the left (balanced) subtree
- All the crs after the middle element must be placed in the right (balanced) subtree
- The middle element is indexed by (quotient (length a-locr) 2)
- How then are the elements before and after the middle element referenced?
- The elements before the middle element are indexed by the interval [0..(sub1 (quotient (length a-locr) 2))]
- The elements after the middle element are indexed by the interval [(add1 (quotient (length a-locr) 2))..(sub1 (length a-locr))]

# Binary Trees

## Creating a Balanced Binary Search Tree

- Can a balanced binary search tree be created in the first place?
- Assume that we have a (listof cr) sorted in nondecreasing order by id number
- The middle element of the list must be the root of the balanced binary search tree
- All the crs before the middle element must be placed in the left (balanced) subtree
- All the crs after the middle element must be placed in the right (balanced) subtree
- The middle element is indexed by (quotient (length a-locr) 2)
- How then are the elements before and after the middle element referenced?
- The elements before the middle element are indexed by the interval [0..(sub1 (quotient (length a-locr) 2))]
- The elements after the middle element are indexed by the interval [(add1 (quotient (length a-locr) 2))..(sub1 (length a-locr))]
- Suggests that we can create a balanced binary search tree using a function that processes an interval
- If the given (listof cr) is empty then the interval to process must be empty
- Otherwise, the interval to process is defined by the minimum and the maximum indexes into the list

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Binary Trees

## Creating a Balanced Binary Search Tree

- (define SORTED-LOCR (list PAULIE SCARFACE VITO BABYFACE TEFLONDON DILLINGER BUGSY))

# Binary Trees

## Creating a Balanced Binary Search Tree

- ```
(define SORTED-LOCR (list PAULIE SCARFACE VITO BABYFACE
                           TEFLONDON DILLINGER BUGSY))
```

- ; Sample expressions for locr->bstocr

```
(define LOCR->BBST1 (create-bstofcr CR-DB0 0 -1))
  (define LOCR->BBST2 (create-bstofcr SORTED-LOCR 0 6))
```

Binary Trees

Creating a Balanced Binary Search Tree

- ```
(define SORTED-LOCR (list PAULIE SCARFACE VITO BABYFACE
 TEFLONDON DILLINGER BUGSY))
```
- ```
;; (listof cr) → (bbstof cr)
;; Purpose: Create (bbst cr) from the given sorted (listof cr)
;; ASSUMPTION: The given (listof cr) is sorted in nondecreasing
;;             order by id number
(define (locr->bstocr a-locr)
```
- ; Sample expressions for locr->bstocr

```
(define LOCR->BBST1 (create-bstofcr CR-DB0 0 -1))
(define LOCR->BBST2 (create-bstofcr SORTED-LOCR 0 6))
```

Binary Trees

Creating a Balanced Binary Search Tree

- (define SORTED-LOCR (list PAULIE SCARFACE VITO BABYFACE TEFLONDON DILLINGER BUGSY))
 - ;; (listof cr) → (bbstof cr)
;; Purpose: Create (bbst cr) from the given sorted (listof cr)
;; ASSUMPTION: The given (listof cr) is sorted in nondecreasing
;; order by id number
(define (locr->bbstocr a-locr)
 - ; Sample expressions for locr->bstocr
(define LOCR->BBST1 (create-bstofcr CR-DB0 0 -1))
(define LOCR->BBST2 (create-bstofcr SORTED-LOCR 0 6))
 - ;; Tests using sample computations for locr->bstocr
(check-expect (locr->bbstocr CR-DB0) LOCR->BBST1)
(check-expect (locr->bbstocr SORTED-LOCR) LOCR->BBST2)
- ;; Tests using sample values for locr->bstocr
(check-expect (locr->bbstocr (list PAULIE VITO BUGSY))
 (list VITO
 (list PAULIE '() '())
 (list BUGSY '() '()))))

Binary Trees

Creating a Balanced Binary Search Tree

- (define SORTED-LOCR (list PAULIE SCARFACE VITO BABYFACE TEFLONDON DILLINGER BUGSY))
- ;; (listof cr) → (bbstof cr)
;; Purpose: Create (bbst cr) from the given sorted (listof cr)
;; ASSUMPTION: The given (listof cr) is sorted in nondecreasing
;; order by id number
(define (locr->bstocr a-locr))
- (create-bbstofcr a-locr 0 (sub1 (length a-locr))))
- ; Sample expressions for locr->bstocr
(define LOCR->BBST1 (create-bstofcr CR-DB0 0 -1))
(define LOCR->BBST2 (create-bstofcr SORTED-LOCR 0 6))
- ;; Tests using sample computations for locr->bstocr
(check-expect (locr->bstocr CR-DB0) LOCR->BBST1)
(check-expect (locr->bstocr SORTED-LOCR) LOCR->BBST2)

;; Tests using sample values for locr->bstocr
(check-expect (locr->bstocr (list PAULIE VITO BUGSY))
 (list VITO
 (list PAULIE '() '())
 (list BUGSY '() '()))))

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

Creating a Balanced Binary Search Tree

- `create-bbstofcr` has as input a (`listof cr`) and an interval of indexes, `[low..high]`, into the list

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

Creating a Balanced Binary Search Tree

- `create-bbstofcr` has as input a (`listof cr`) and an interval of indexes, `[low..high]`, into the list
- Reason about the structure of the interval

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

Creating a Balanced Binary Search Tree

- `create-bbstofcr` has as input a (`listof cr`) and an interval of indexes, `[low..high]`, into the list
- Reason about the structure of the interval
- If the interval is empty then the needed tree is `'()`

Binary Trees

Creating a Balanced Binary Search Tree

- `create-bbstofcr` has as input a (`listof cr`) and an interval of indexes, `[low..high]`, into the list
- Reason about the structure of the interval
- If the interval is empty then the needed tree is `'()`
- If the tree is not empty then the root of the needed tree is the middle list element in the interval:

```
(+ low (quotient (- high low) 2))
```

Binary Trees

Creating a Balanced Binary Search Tree

- `create-bbstofcr` has as input a (`listof cr`) and an interval of indexes, `[low..high]`, into the list
- Reason about the structure of the interval
- If the interval is empty then the needed tree is `'()`
- If the tree is not empty then the root of the needed tree is the middle list element in the interval:

`(+ low (quotient (- high low) 2))`

- The elements for the left and right subtrees:

`[low..(sub1 (+ low (quotient (- high low) 2)))]`

`[(add1 (+ low (quotient (- high low) 2)))..high]`

Binary Trees

Creating a Balanced Binary Search Tree

- ;; Sample expressions for create-bbstofcr

```
(define CBST1 '())  
(define CBST2  
  (list (list-ref SORTED-LOCR (+ 0 (quotient (- 6 0) 2)))  
        (create-bbstofcr  
          SORTED-LOCR 0 (sub1 (+ 0 (quotient (- 6 0) 2))))  
        (create-bbstofcr  
          SORTED-LOCR (add1 (+ 0 (quotient (- 6 0) 2))) 6)))
```

Binary Trees

Creating a Balanced Binary Search Tree

- ```
;; (listof cr) [natnum..natnum] → (bbstof cr)
;; Purpose: Create a bbst from the sorted list in given interval
(define (create-bbstofcr a-locr low high)
```
- ```
;; Sample expressions for create-bbstofcr
(define CBST1 '())
(define CBST2
  (list (list-ref SORTED-LOCR (+ 0 (quotient (- 6 0) 2)))
        (create-bbstofcr
          SORTED-LOCR 0 (sub1 (+ 0 (quotient (- 6 0) 2))))
        (create-bbstofcr
          SORTED-LOCR (add1 (+ 0 (quotient (- 6 0) 2))) 6)))
```

Binary Trees

Creating a Balanced Binary Search Tree

- ```
;; (listof cr) [natnum..natnum] → (bbstof cr)
;; Purpose: Create a bbst from the sorted list in given interval
(define (create-bbstofcr a-locr low high)
```
- ```
; Sample expressions for create-bbstofcr
(define CBST1 '())
(define CBST2
  (list (list-ref SORTED-LOCR (+ 0 (quotient (- 6 0) 2)))
        (create-bbstofcr
          SORTED-LOCR 0 (sub1 (+ 0 (quotient (- 6 0) 2))))
        (create-bbstofcr
          SORTED-LOCR (add1 (+ 0 (quotient (- 6 0) 2))) 6)))
(define CBST3
  (list (list-ref SORTED-LOCR (+ 0 (quotient (- 6 0) 2)))
        (create-bbstofcr
          SORTED-LOCR 0 (sub1 (+ 0 (quotient (- 6 0) 2))))
        (create-bbstofcr
          SORTED-LOCR (add1 (+ 0 (quotient (- 6 0) 2))) 6))
        (list (list-ref SORTED-LOCR (+ 0 (quotient (- 6 0) 2)))
              (create-bbstofcr
                SORTED-LOCR 0 (sub1 (+ 0 (quotient (- 6 0) 2))))
              (create-bbstofcr
                SORTED-LOCR (add1 (+ 0 (quotient (- 6 0) 2))) 6))))
```
- ```
; Tests using sample computations for create-bstofcr
(check-expect (create-bbstofcr CR-DB0 0 -1) CBST1)
(check-expect (create-bbstofcr SORTED-LOCR 0 6) CBST2)
;; Tests using sample values for create-bstofcr
(check-expect
 (create-bbstofcr
 (list PAULIE SCARFACE VITO DILLINGER BUGSY) 0 4)
 (list VITO
 (list PAULIE '() (list SCARFACE '() '()))
 (list DILLINGER '() (list BUGSY '() '()))))
```

# Binary Trees

## Creating a Balanced Binary Search Tree

- ```
;; (listof cr) [natnum..natnum] → (bbstof cr)
;; Purpose: Create a bbst from the sorted list in given interval
(define (create-bbstofcr a-locr low high)
  (if (< high low)
      '()
      (list (list-ref a-locr (+ low (quotient (- high low) 2)))
            (create-bbstofcr
              a-locr low (sub1 (+ low (quotient (- high low) 2))))
            (create-bbstofcr
              a-locr (add1 (+ low (quotient (- high low) 2))) high))))
```
- ```
;; Sample expressions for create-bbstofcr
(define CBST1 '())
(define CBST2
 (list (list-ref SORTED-LOCR (+ 0 (quotient (- 6 0) 2)))
 (create-bbstofcr
 SORTED-LOCR 0 (sub1 (+ 0 (quotient (- 6 0) 2))))
 (create-bbstofcr
 SORTED-LOCR (add1 (+ 0 (quotient (- 6 0) 2))) 6)))
```
- ```
;; Tests using sample computations for create-bstofcr
(check-expect (create-bbstofcr CR-DB0 0 -1) CBST1)
(check-expect (create-bbstofcr SORTED-LOCR 0 6) CBST2)
;; Tests using sample values for create-bstofcr
(check-expect
  (create-bbstofcr
    (list PAULIE SCARFACE VITO DILLINGER BUGSY) 0 4)
  (list VITO
    (list PAULIE '() (list SCARFACE '() '()))
    (list DILLINGER '() (list BUGSY '() '()))))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

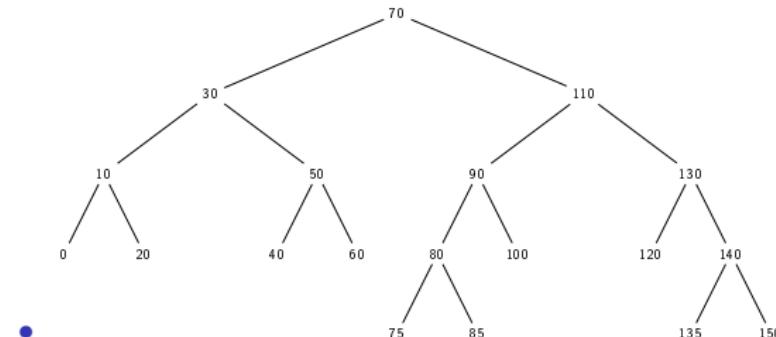
Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?

Binary Trees

Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?

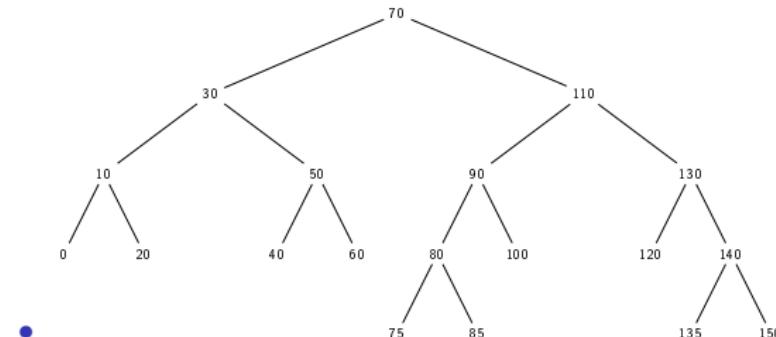


- Consider searching for 87

Binary Trees

Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?

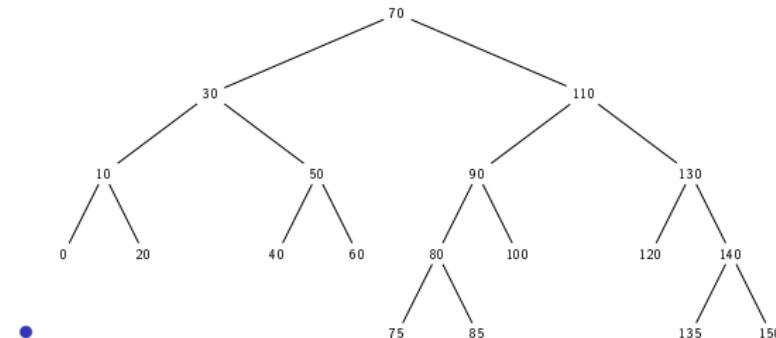


- Consider searching for 87
- After comparing 87 with 70 all the elements in the left subtree are discarded from the search

Binary Trees

Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?

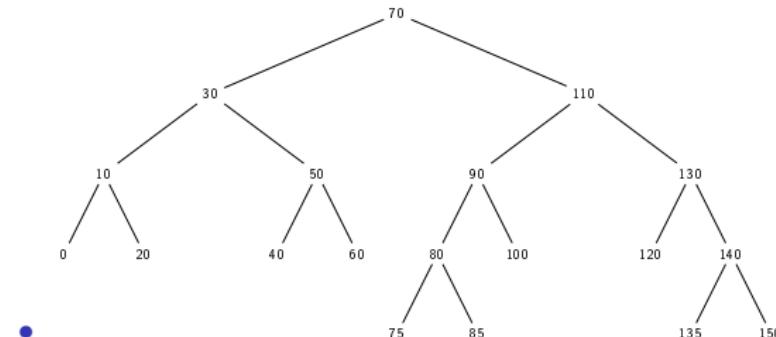


- Consider searching for 87
- After comparing 87 with 70 all the elements in the left subtree are discarded from the search
- After the comparison with 110 half of the remaining elements are discarded again from the search.

Binary Trees

Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?

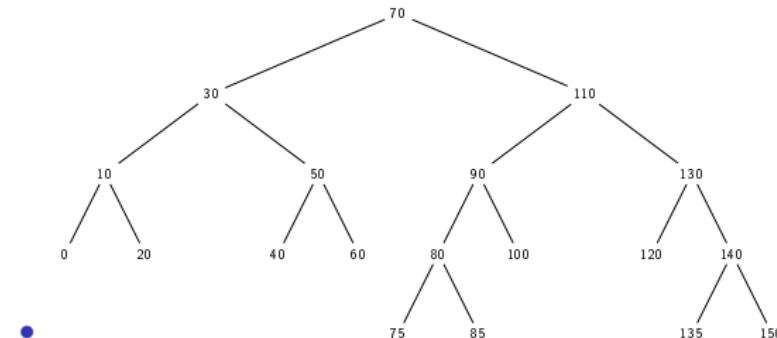


- Consider searching for 87
- After comparing 87 with 70 all the elements in the left subtree are discarded from the search
- After the comparison with 110 half of the remaining elements are discarded again from the search.
- This happens again after the comparison with 90 and with 80

Binary Trees

Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?

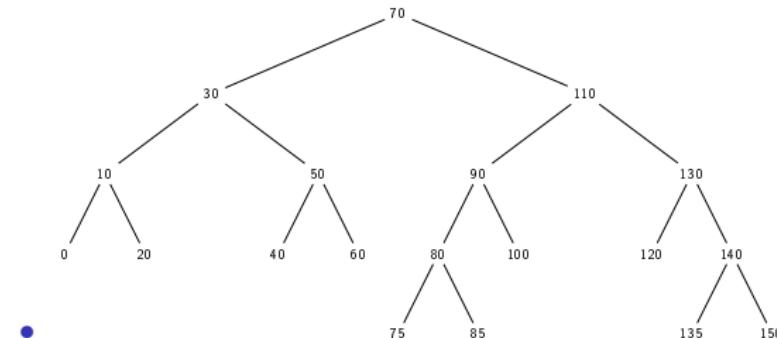


- Consider searching for 87
- After comparing 87 with 70 all the elements in the left subtree are discarded from the search
- After the comparison with 110 half of the remaining elements are discarded again from the search.
- This happens again after the comparison with 90 and with 80
- Finally, after the comparison with 85 `get-record-bbstocr` is called with the empty tree and the function halts

Binary Trees

Creating a Balanced Binary Search Tree

- Does representing a database as a balanced binary search tree improve the complexity of searching?



- Consider searching for 87
- After comparing 87 with 70 all the elements in the left subtree are discarded from the search
- After the comparison with 110 half of the remaining elements are discarded again from the search.
- This happens again after the comparison with 90 and with 80
- Finally, after the comparison with 85 `get-record-bbstocr` is called with the empty tree and the function halts
- Notice that 6 calls are made to `get-record-bbstocr`

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
70 → 110 → 130 → 140 → 135
- What is the length of the longest path in a balanced binary search tree with n nodes?

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
 $70 \rightarrow 110 \rightarrow 130 \rightarrow 140 \rightarrow 135$
- What is the length of the longest path in a balanced binary search tree with n nodes?
- After each step down the tree about half of the elements are discarded
- How many times can we divide n by 2?

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
 $70 \rightarrow 110 \rightarrow 130 \rightarrow 140 \rightarrow 135$
- What is the length of the longest path in a balanced binary search tree with n nodes?
- After each step down the tree about half of the elements are discarded
- How many times can we divide n by 2?
- If n is a power of 2 then the answer is $\lg n$ (i.e., $\log_2 n$)

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
 $70 \rightarrow 110 \rightarrow 130 \rightarrow 140 \rightarrow 135$
- What is the length of the longest path in a balanced binary search tree with n nodes?
- After each step down the tree about half of the elements are discarded
- How many times can we divide n by 2?
- If n is a power of 2 then the answer is $\lg n$ (i.e., $\log_2 n$)
- If n is not a power of 2 then the answer is $\lfloor \lg n \rfloor$, the largest integer less than or equal to $\lg n$

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
 $70 \rightarrow 110 \rightarrow 130 \rightarrow 140 \rightarrow 135$
- What is the length of the longest path in a balanced binary search tree with n nodes?
- After each step down the tree about half of the elements are discarded
- How many times can we divide n by 2?
- If n is a power of 2 then the answer is $\lg n$ (i.e., $\log_2 n$)
- If n is not a power of 2 then the answer is $\lfloor \lg n \rfloor$, the largest integer less than or equal to $\lg n$
- This means that in the worst case $\lfloor \lg n \rfloor + 2$ calls are made
- The complexity of searching a balanced binary search tree is $O(\lg n)$

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
 $70 \rightarrow 110 \rightarrow 130 \rightarrow 140 \rightarrow 135$
- What is the length of the longest path in a balanced binary search tree with n nodes?
- After each step down the tree about half of the elements are discarded
- How many times can we divide n by 2?
- If n is a power of 2 then the answer is $\lg n$ (i.e., $\log_2 n$)
- If n is not a power of 2 then the answer is $\lfloor \lg n \rfloor$, the largest integer less than or equal to $\lg n$
- This means that in the worst case $\lfloor \lg n \rfloor + 2$ calls are made
- The complexity of searching a balanced binary search tree is $O(\lg n)$
- This is significantly better than an $O(n)$.
- In the worst case, the number of function calls to search a database with 1024 crs represented as a (listof cr) is proportional to 1024
- The number of calls is proportional to 10 when the database is represented as a balanced binary search tree.

Binary Trees

Creating a Balanced Binary Search Tree

- In the worst case the number of calls is equal to the height of the tree plus 2
- Such is the case, for example, for this path:
 $70 \rightarrow 110 \rightarrow 130 \rightarrow 140 \rightarrow 135$
- What is the length of the longest path in a balanced binary search tree with n nodes?
- After each step down the tree about half of the elements are discarded
- How many times can we divide n by 2?
- If n is a power of 2 then the answer is $\lg n$ (i.e., $\log_2 n$)
- If n is not a power of 2 then the answer is $\lfloor \lg n \rfloor$, the largest integer less than or equal to $\lg n$
- This means that in the worst case $\lfloor \lg n \rfloor + 2$ calls are made
- The complexity of searching a balanced binary search tree is $O(\lg n)$
- This is significantly better than an $O(n)$.
- In the worst case, the number of function calls to search a database with 1024 crs represented as a (listof cr) is proportional to 1024
- The number of calls is proportional to 10 when the database is represented as a balanced binary search tree.
- Equally noteworthy is that `create-bstofcr` is a recursive function that is not based on structural recursion
- Recursive calls not made with the subinterval used to build the interval
- New intervals are generated for recursive calls: *generative recursion*
- We shall study generative recursion more in-depth in the next volume of this textbook

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Binary Trees

Homework

- **Problems:** 186–188, 191–192

Mutually Recursive Data

- We have only considered data types that only refer to themselves:
A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

- We have only considered data types that only refer to themselves:
A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)
- Now, take a look at the data definition for a binary tree:
;; A binary tree of X, (btot X), is either:
;; 1. '()
;; 2. (list X (btot X) (btot X))
- This data definition also only refers to itself

Mutually Recursive Data

- We have only considered data types that only refer to themselves:
A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)
- Now, take a look at the data definition for a binary tree:

```
;; A binary tree of X, (btot X), is either:  
;;   1. '()  
;;   2. (list X (btot X) (btot X))
```
- This data definition also only refers to itself
- A non-empty binary tree is not of arbitrary size: always has 3 elements
- A non-empty binary tree ought to be represented using a structure

Mutually Recursive Data

- We have only considered data types that only refer to themselves:
A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)
- Now, take a look at the data definition for a binary tree:
;; A binary tree of X, (btot X), is either:
;; 1. '()
;; 2. (list X (btot X) (btot X))
- This data definition also only refers to itself
- A non-empty binary tree is not of arbitrary size: always has 3 elements
- A non-empty binary tree ought to be represented using a structure
- A first attempt:

```
;; A binary tree of X, (btot X), is either:  
;;     1. '()  
;;     2. A structure, (make-node X (btot X) (btot X)), with  
;;         an X value and two subtrees of X values
```

Mutually Recursive Data

- We have only considered data types that only refer to themselves:
A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)
- Now, take a look at the data definition for a binary tree:

```
;; A binary tree of X, (btot X), is either:  
;;   1. '()  
;;   2. (list X (btot X) (btot X))
```

- This data definition also only refers to itself
- A non-empty binary tree is not of arbitrary size: always has 3 elements
- A non-empty binary tree ought to be represented using a structure
- A first attempt:

```
;; A binary tree of X, (btot X), is either:  
;;   1. '()  
;;   2. A structure, (make-node X (btot X) (btot X)), with  
;;      an X value and two subtrees of X values
```

- We have a data definition within a data definition

Mutually Recursive Data

- We have only considered data types that only refer to themselves:

A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)

- Now, take a look at the data definition for a binary tree:

```
;; A binary tree of X, (btot X), is either:  
    ;; 1. '()  
    ;; 2. (list X (btot X) (btot X))
```

- This data definition also only refers to itself
- A non-empty binary tree is not of arbitrary size: always has 3 elements
- A non-empty binary tree ought to be represented using a structure
- A first attempt:

```
;; A binary tree of X, (btot X), is either:  
    ;; 1. '()  
    ;; 2. A structure, (make-node X (btot X) (btot X)), with  
        ;; an X value and two subtrees of X values
```

- We have a data definition within a data definition
- More natural to have distinct and separate data definitions:

```
;; A (nodeof X) is a structure, (make-node X (btot X) (btot X)),  
;; with an X value and two (btot X) values.  
;; A binary tree of X, (btot X), is either:  
    ;; 1. '()  
    ;; 2. (nodeof X)
```

Mutually Recursive Data

- We have only considered data types that only refer to themselves:
A (listof X) is either: A natural number (natnum) is either:
1. '() 2. (cons X (listof X)) 1. 0 2. (add1 natnum)
- Now, take a look at the data definition for a binary tree:

```
;; A binary tree of X, (btot X), is either:  
    ;; 1. '()  
    ;; 2. (list X (btot X) (btot X))
```
- This data definition also only refers to itself
- A non-empty binary tree is not of arbitrary size: always has 3 elements
- A non-empty binary tree ought to be represented using a structure
- A first attempt:

```
;; A binary tree of X, (btot X), is either:  
    ;; 1. '()  
    ;; 2. A structure, (make-node X (btot X) (btot X)), with  
        ;; an X value and two subtrees of X values
```
- We have a data definition within a data definition
- More natural to have distinct and separate data definitions:

```
;; A (nodeof X) is a structure, (make-node X (btot X) (btot X)),  
;; with an X value and two (btot X) values.  
;; A binary tree of X, (btot X), is either:  
    ;; 1. '()  
    ;; 2. (nodeof X)
```
- These data definitions are *mutually recursive*
- They define intertwined data that refer to each other

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Designing with Mutually Recursive Data

- What impact does mutually recursive data have on problem solving?
- In a very real sense we already know what references to a data definition mean for our code.
- A selfreference turns into a recursive call in our code

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Designing with Mutually Recursive Data

- What impact does mutually recursive data have on problem solving?
- In a very real sense we already know what references to a data definition mean for our code.
- A selfreference turns into a recursive call in our code
- The same principle applies to mutually recursive data definitions
- The reference to a node in the data definition for a (btot X) means that processing a non-empty binary tree requires calling a function on a node
- Similarly, the references to a (btot X) in the data definition of a node require calls to a function that processes a (btot X)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Designing with Mutually Recursive Data

- What impact does mutually recursive data have on problem solving?
- In a very real sense we already know what references to a data definition mean for our code.
- A selfreference turns into a recursive call in our code
- The same principle applies to mutually recursive data definitions
- The reference to a node in the data definition for a (btot X) means that processing a non-empty binary tree requires calling a function on a node
- Similarly, the references to a (btot X) in the data definition of a node require calls to a function that processes a (btot X)
- Observe that the data definition for a (btot X) is still circular
- A (btot X) refers to node and a node refers (back) to (btot X)
- It defines data of arbitrary size
- The definition is useful because the circularity ends when a (btot X) is '()

Mutually Recursive Data

Designing with Mutually Recursive Data

- #| ;; Sample instances of (nodeof X)
(define NODE0 (make-node)) . . .
;; node . . . → . . . Purpose:
(define (f-on-node a-node)
 . . .(f-on-X (node-val a-node))
 . . .(f-on-btx (node-ltree a-node))
 . . .(f-on-btx (node-rtree a-node)))
;; Sample expressions for f-on-node
(define NODE0-VAL . . .) . . .
;; Tests using sample computations for f-on-node
(check-expect (f-on-node NODE0 . . .) NODE0-VAL) . . .
;; Tests using sample values for f-on-node
(check-expect (f-on-node (make-node) . . .) . . .) . . .

Mutually Recursive Data

Designing with Mutually Recursive Data

- #| ; Sample instances of (nodeof X)
(define NODE0 (make-node ...)) ...
;; node ... → ... Purpose:
(define (f-on-node a-node)
 ... (f-on-X (node-val a-node))
 ... (f-on-btx (node-ltree a-node))
 ... (f-on-btx (node-rtree a-node)))
;; Sample expressions for f-on-node
(define NODE0-VAL ...) ...
;; Tests using sample computations for f-on-node
(check-expect (f-on-node NODE0 ...) NODE0-VAL) ...
;; Tests using sample values for f-on-node
(check-expect (f-on-node (make-node ...)) ...) ...) ...
- ; Sample instances of (bttx X)
(define BTX0 '()) (define BTX1 (make-node ...)) ...
;; btx ... → ... Purpose:
(define (f-on-btx a-btx ...)
 (if (empty? a-node)
 ...
 (f-on-node a-btx ...)))
;; Sample expressions for f-on-btx
(define BTX0-VAL (f-on-btx BTX0 ...))
(define BTX1-VAL (f-on-node BTX1 ...)) ...
;; Tests using sample computations for f-on-btx
(check-expect (f-on-btx BTX0 ...) BTX0-VAL)
(check-expect (f-on-btx BTX1 ...) BTX1-VAL)
...
;; Tests using sample values for f-on-btx
(check-expect (f-on-btx ...) ...) ...)#
(define-struct node (val ltree rtree))

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

- We revisit this problem and solve it based on our refined data definition
- We define sample instances of (nodeof int) and (btot int)
- Care must be taken writing these in our program because the definitions are mutually recursive
- We may not define a binary tree that only contains 177 as follows:

```
(define NODE177 (make-node 177 BTIO BTIO))
```

```
(define BTIO '())
```

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

- We revisit this problem and solve it based on our refined data definition
- We define sample instances of (nodeof int) and (btot int)
- Care must be taken writing these in our program because the definitions are mutually recursive
- We may not define a binary tree that only contains 177 as follows:

```
(define NODE177 (make-node 177 BTIO BTIO))
```

```
(define BTIO '())
```

- The proper way to build this tree is:

```
(define BTIO '())
```

```
(define NODE177 (make-node 177 BTIO BTIO))
```

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

- We revisit this problem and solve it based on our refined data definition
- We define sample instances of (nodeof int) and (btot int)
- Care must be taken writing these in our program because the definitions are mutually recursive
- We may not define a binary tree that only contains 177 as follows:

```
(define NODE177 (make-node 177 BTIO BTIO))
```

```
(define BTIO '())
```

- The proper way to build this tree is:

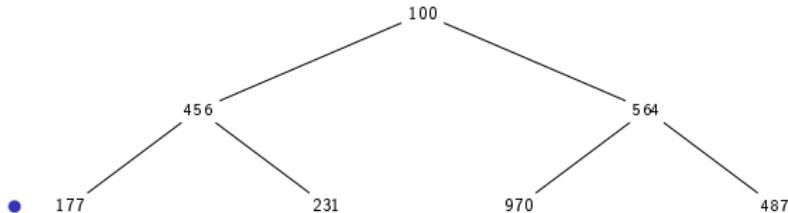
```
(define BTIO '())
```

```
(define NODE177 (make-node 177 BTIO BTIO))
```

- This indicates the general strategy that we must follow to build instances of mutually recursive data
- Build non-recursive instances first and then build instances that only depend on existing instances

Mutually Recursive Data

Revisiting the Maximum of a (bt of int)



Lists

List Processing

Natural Numbers

Interval Processing

Aliens Attack Version 4

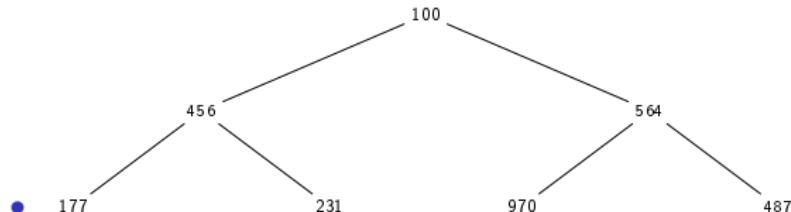
Binary Trees

Mutually Recursive Data

Processing Multiple Inputs of Arbitrary Size

Mutually Recursive Data

Revisiting the Maximum of a (btot int)



- `(define BTIO '())`

```
(define NODE177 (make-node 177 BTIO BTIO))
(define NODE231 (make-node 231 BTIO BTIO))
(define NODE970 (make-node 970 BTIO BTIO))
(define NODE487 (make-node 487 BTIO BTIO))
(define BTI177 NODE177)  (define BTI231 NODE231)
(define BTI970 NODE970)  (define BTI487 NODE487)

(define NODE456 (make-node 456 BTI177 BTI231))
(define NODE564 (make-node 564 BTI970 BTI487))
(define BTI456 NODE456)  (define BTI564 NODE564)

(define NODE100 (make-node 100 BTI456 BTI564))

(define BTI100 NODE100)
```

Mutually Recursive Data

Revisiting the Maximum of a (btint int)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ;; Sample expressions for max-btint
(define BTI100-VAL (max-node BTI100))
(define BTI456-VAL (max-node BTI456))

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

- `;; (btot int) → int throws error`
`;; Purpose: Return the maximum of the given (btot int)`
`(define (max-btint a-btint)`
- `;; Sample expressions for max-btint`
`(define BTI100-VAL (max-node BTI100))`
`(define BTI456-VAL (max-node BTI456))`

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

- `;; (btot int) → int throws error`
`;; Purpose: Return the maximum of the given (btot int)`
`(define (max-btint a-btint)`

- `;; Sample expressions for max-btint`
`(define BTI100-VAL (max-node BTI100))`
`(define BTI456-VAL (max-node BTI456))`
- `;; Tests using sample computations for max-btint`
`(check-expect (max-btint BTI100) BTI100-VAL)`
`(check-expect (max-btint BTI456) BTI456-VAL)`

```
; Tests using sample values for f-on-btx
(check-error
  (max-btint BTI0)
  "An empty (btot int) has no maximum value.")
(check-expect (max-btint (make-node -97
                                         (make-node -89 BTI0 BTI0)
                                         (make-node -99 BTI0 BTI0))))
```

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

- `;; (btot int) → int throws error`
`;; Purpose: Return the maximum of the given (btot int)`
`(define (max-btint a-btint))`
- `(if (empty? a-btint)`
 `(error "An empty (btot int) has no maximum value.")`
 `(max-node a-btint)))`
- `;; Sample expressions for max-btint`
`(define BTI100-VAL (max-node BTI100))`
`(define BTI456-VAL (max-node BTI456))`
- `;; Tests using sample computations for max-btint`
`(check-expect (max-btint BTI100) BTI100-VAL)`
`(check-expect (max-btint BTI456) BTI456-VAL)`

`; Tests using sample values for f-on-btx`
`(check-error`
 `(max-btint BTI0)`
 `"An empty (btot int) has no maximum value.")`
`(check-expect (max-btint (make-node -97`
 `(make-node -89 BTI0 BTI0)`
 `(make-node -99 BTI0 BTI0))))`

Mutually Recursive Data

Revisiting the Maximum of a (btot int)

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ```
(define NODE200 (make-node 200 BT10 (make-node 311 BT10 BT10)))
(define NODE650 (make-node 650 (make-node 912 BT10 BT10) BT10))
;; Sample expressions for max-node
(define NODE487-VAL (node-val NODE487))
(define NODE200-VAL (max (node-val NODE200) (max-node (node-rtree NODE200))))
(define NODE650-VAL (max (node-val NODE650) (max-node (node-ltree NODE650))))
(define NODE456-VAL (max (node-val NODE456)
 (max-node (node-ltree NODE456))
 (max-node (node-rtree NODE456))))
```

# Mutually Recursive Data

## Revisiting the Maximum of a (btot int)

- ```
;; node → int
;; Purpose: Return the max int in the given node
(define (max-node a-node)
```

- ```
(define NODE200 (make-node 200 BT10 (make-node 311 BT10 BT10)))
(define NODE650 (make-node 650 (make-node 912 BT10 BT10) BT10))
;; Sample expressions for max-node
(define NODE487-VAL (node-val NODE487))
(define NODE200-VAL (max (node-val NODE200) (max-node (node-rtree NODE200))))
(define NODE650-VAL (max (node-val NODE650) (max-node (node-ltree NODE650))))
(define NODE456-VAL (max (node-val NODE456)
 (max-node (node-ltree NODE456))
 (max-node (node-rtree NODE456))))
```

# Mutually Recursive Data

## Revisiting the Maximum of a (btot int)

- ```
;; node → int
;; Purpose: Return the max int in the given node
(define (max-node a-node)
```

- ```
(define NODE200 (make-node 200 BT10 (make-node 311 BT10 BT10)))
(define NODE650 (make-node 650 (make-node 912 BT10 BT10) BT10))
;; Sample expressions for max-node
(define NODE487-VAL (node-val NODE487))
(define NODE200-VAL (max (node-val NODE200) (max-node (node-rtree NODE200))))
(define NODE650-VAL (max (node-val NODE650) (max-node (node-ltree NODE650))))
(define NODE456-VAL (max (node-val NODE456)
 (max-node (node-ltree NODE456))
 (max-node (node-rtree NODE456))))
;; Tests using sample computations for max-node
(check-expect (max-node NODE487) NODE487-VAL)
(check-expect (max-node NODE200) NODE200-VAL)
(check-expect (max-node NODE650) NODE650-VAL)
(check-expect (max-node NODE456) NODE456-VAL)
;; Tests using sample values for max-node
(check-expect (max-node (make-node 789
 (make-node -1000 BT10 BT10)
 (make-node 3000 BT10 BT10)))
```

# Mutually Recursive Data

## Revisiting the Maximum of a (bt of int)

- ```
;; node → int
;; Purpose: Return the max int in the given node
(define (max-node a-node)
  (cond [(and (empty? (node-ltree a-node))
              (empty? (node-rtree a-node)))
          (node-val a-node)]
        [(empty? (node-ltree a-node))
         (max (node-val a-node) (max-btint (node-rtree a-node)))]
        [(empty? (node-rtree a-node))
         (max (node-val a-node) (max-btint (node-ltree a-node)))]
        [else (max (node-val a-node)
                    (max-btint (node-ltree a-node))
                    (max-btint (node-rtree a-node))))]))
```
- ```
(define NODE200 (make-node 200 BT10 (make-node 311 BT10 BT10)))
(define NODE650 (make-node 650 (make-node 912 BT10 BT10) BT10))
;; Sample expressions for max-node
(define NODE487-VAL (node-val NODE487))
(define NODE200-VAL (max (node-val NODE200) (max-node (node-rtree NODE200))))
(define NODE650-VAL (max (node-val NODE650) (max-node (node-ltree NODE650))))
(define NODE456-VAL (max (node-val NODE456)
 (max-node (node-ltree NODE456))
 (max-node (node-rtree NODE456))))
```
- ```
; Tests using sample computations for max-node
(check-expect (max-node NODE487) NODE487-VAL)
(check-expect (max-node NODE200) NODE200-VAL)
(check-expect (max-node NODE650) NODE650-VAL)
(check-expect (max-node NODE456) NODE456-VAL)
;; Tests using sample values for max-node
(check-expect (max-node (make-node 789
                                    (make-node -1000 BT10 BT10)
                                    (make-node 3000 BT10 BT10)))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Evaluating Arithmetic Expressions

- We tackle the problem of evaluating the subset of symbolic expressions representing arithmetic expressions using +, -, and *

Mutually Recursive Data

Evaluating Arithmetic Expressions

- We tackle the problem of evaluating the subset of symbolic expressions representing arithmetic expressions using +, -, and *
- To start, let's look at a few sample expressions:

45

(+ 67 54)

87

(* (- 56 43) 44 (+ 7 4))

- There are two varieties of symbolic expressions
- The simplest symbolic expression is a number
- The second is a list that contains a symbol representing a function followed by a list of symbolic expressions for the arguments

Mutually Recursive Data

Evaluating Arithmetic Expressions

- We tackle the problem of evaluating the subset of symbolic expressions representing arithmetic expressions using +, -, and *
- To start, let's look at a few sample expressions:

45 87
(+ 67 54) (* (- 56 43) 44 (+ 7 4))

- There are two varieties of symbolic expressions
- The simplest symbolic expression is a number
- The second is a list that contains a symbol representing a function followed by a list of symbolic expressions for the arguments
- Data definitions:

A symbolic expression (sexpr) is either:

1. number
2. slist

A symbolic list (slist) is: (cons function loexpr)

A function is either:

1. '+
2. '-
3. '*

A list of sexpr (loexpr) is either:

1. '()
2. (cons sexpr loexpr)

Mutually Recursive Data

Evaluating Arithmetic Expressions

- #1 ;; Samples of sexpr
(define SEXP1 ...) (define SEXP2 ...)
;; sexpr ... → ... Purpose:
(define (f-on-sexpr a-sexpr)
 (if (number? a-sexpr)
 (f-on-number a-sexpr ...)
 (f-on-slist a-sexpr ...)))
;; Sample expressions for f-on-sexpr
(define SEXP1-VAL ...) (define SEXP2-VAL ...)

;; Tests using sample computations for f-on-sexpr
(check-expect (f-on-sexpr SEXP1) SEXP1-VAL)
(check-expect (f-on-sexpr SEXP2) SEXP2-VAL) ...
;; Tests using sample values for f-on-sexpr
(check-expect (f-on-sexpr ...) ...)

Mutually Recursive Data

Evaluating Arithmetic Expressions

- #| ;; Samples of sexpr
(define SEXP1 ...) (define SEXP2 ...)
;; sexpr ... → ... Purpose:
(define (f-on-sexpr a-sexpr)
 (if (number? a-sexpr)
 (f-on-number a-sexpr ...)
 (f-on-slist a-sexpr ...)))
;; Sample expressions for f-on-sexpr
(define SEXP1-VAL ...) (define SEXP2-VAL ...)

;; Tests using sample computations for f-on-sexpr
(check-expect (f-on-sexpr SEXP1) SEXP1-VAL)
(check-expect (f-on-sexpr SEXP2) SEXP2-VAL) ...
;; Tests using sample values for f-on-sexpr
(check-expect (f-on-sexpr ...) ...) ...
...
• ;; Samples of slist
(define SLIST1 '()) (define SLIST2 ...)

;; slist ... → ... Purpose:
(define (f-on-slist an-slist)
 (if (empty? an-slist)
 ...
 (cons (f-on-sexpr (first an-slist) ...)
 (f-on-slist (rest an-slist) ...))))
;; Sample expressions for f-on-slist
(define SLIST1-VAL ...) (define SLIST2-VAL ...)
;; Tests using sample computations for f-on-slist
(check-expect (f-on-slist SLIST1 ...) SLIST1-VAL)
(check-expect (f-on-slist SLIST2 ...) SLIST2-VAL) ...
;; Tests using sample values for f-on-slist
(check-expect (f-on-slist ...) ...) ...
...|#

Mutually Recursive Data

Evaluating Arithmetic Expressions

- #| ;; function ... → ... Purpose:

```
(define (f-on-function a-function)
  (cond [(eq? a-function '+) ...]
        [(eq? a-function '-) ...]
        [else ...]))
```


;; Sample expressions for f-on-function

```
(define PLUS-VAL ...)
(define SUBT-VAL ...)
(define MULT-VAL ...)
```


;; Tests using sample computations for f-on-function

```
(check-expect (f-on-function PLUS ...) PLUS-VAL)
(check-expect (f-on-function SUBT ...) SUBT-VAL)
(check-expect (f-on-function MULT ...) MULT-VAL)
```


;; Tests using sample values for f-on-function

```
(check-expect (f-on-function ... ...) ...)
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- Start by defining sample instances:

```
;; Sample instances of slist
(define SLIST1 '(* (+ 44 -44) (- 20 10)))
(define SLIST2 '(* 3 (+ 6 4)))

;; Sample instances of sexpr
(define SEXP1 67)
(define SEXP2 SLIST2)

;; Sample instances of (listof sexpr)
(define LOSEXP1 '())
(define LOSEXP2 '(80 (+ 70 20) (* 4 (- 30 5)))

;; Sample instances of (listof number)
(define LON1 '())
(define LON2 '(1 2 3 4 5))
(define LON3 '(-10 -20 -30))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ;; Sample expressions for eval-seq
`(define SEXP1-VAL SEXP1) ;; for a number
(define SEXP2-VAL (eval-slist SEXP2)) ;; for a slist`

Mutually Recursive Data

Evaluating Arithmetic Expressions

- `;; sexpr → number throws error Reason for error will be explained`
`;; Purpose: To evaluate the given sexpr`
`(define (eval-sexpr a-sexpr)`
- `;; Sample expressions for eval-sexpr`
`(define SEXP1-VAL SEXP1) ;; for a number`
`(define SEXP2-VAL (eval-slist SEXP2)) ;; for a slist`

Mutually Recursive Data

Evaluating Arithmetic Expressions

- `;; sexpr → number throws error Reason for error will be explained`
`;; Purpose: To evaluate the given sexpr`
`(define (eval-sexpr a-sexpr)`
 - `(if (number? a-sexpr)`
`a-sexpr`
`(eval-slist a-sexpr)))`
 - `;; Sample expressions for eval-sexpr`
`(define SEXP1-VAL SEXP1) ;; for a number`
`(define SEXP2-VAL (eval-slist SEXP2)) ;; for a slist`
 - `;; Tests using sample computations for eval-sexpr`
`(check-expect (eval-sexpr SEXP1) SEXP1-VAL)`
`(check-expect (eval-sexpr SEXP2) SEXP2-VAL)`
- `; Tests using sample values for eval-sexpr`
`(check-expect (eval-sexpr 42) 42)`
`(check-expect (eval-sexpr '(+ -1 (* 6 (+ 8 2)))) 59)`

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; Sample expressions for eval-slist
(define SLIST1-VAL (apply-f (first SLIST1)
 (eval-args (rest SLIST1))))
(define SLIST2-VAL (apply-f (first SLIST2)
 (eval-args (rest SLIST2))))
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; slist → number
;; Purpose: To evaluate the given slist
(define (eval-slist an-slist)
```
- ```
;; Sample expressions for eval-slist
(define SLIST1-VAL (apply-f (first SLIST1)
 (eval-args (rest SLIST1))))
(define SLIST2-VAL (apply-f (first SLIST2)
 (eval-args (rest SLIST2))))
```

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; slist → number
;; Purpose: To evaluate the given slist
(define (eval-slist an-slist)
```
- ```
;; Sample expressions for eval-slist
(define SLIST1-VAL (apply-f (first SLIST1)
 (eval-args (rest SLIST1))))
(define SLIST2-VAL (apply-f (first SLIST2)
 (eval-args (rest SLIST2))))
```
- ```
;; Tests using sample computations for eval-slist
(check-expect (eval-slist SLIST1) SLIST1-VAL)
(check-expect (eval-slist SLIST2) SLIST2-VAL)
```
- ```
;; Tests using sample values for eval-slist
(check-expect (eval-slist
 '(- (* 10 10) (+ 5 (* 2 3) 7))) 82)
```

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; slist → number
;; Purpose: To evaluate the given slist
(define (eval-slist an-slist)
  (apply-f (first an-slist) (eval-args (rest an-slist))))
```
- ```
;; Sample expressions for eval-slist
(define SLIST1-VAL (apply-f (first SLIST1)
 (eval-args (rest SLIST1))))
(define SLIST2-VAL (apply-f (first SLIST2)
 (eval-args (rest SLIST2))))
```
- ```
;; Tests using sample computations for eval-slist
(check-expect (eval-slist SLIST1) SLIST1-VAL)
(check-expect (eval-slist SLIST2) SLIST2-VAL)
```
- ```
;; Tests using sample values for eval-slist
(check-expect (eval-slist
 '(- (* 10 10) (+ 5 (* 2 3) 7))) 82)
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; Sample expressions for eval-args
(define LOSEXPRESS1-VAL '())
(define LOSEXPRESS2-VAL (cons (eval-seexpr (first LOSEXPRESS2))
                               (eval-args (rest LOSEXPRESS2))))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; (listof sexpr) arrow (listof number) throws error
;; Purpose: To evaluate the sexprs in the given list
(define (eval-args a-losexpr)
```
- ```
;; Sample expressions for eval-args
(define LOSEXPRESS1-VAL '())
(define LOSEXPRESS2-VAL (cons (eval-seexpr (first LOSEXPRESS2))
                               (eval-args (rest LOSEXPRESS2))))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; (listof sexpr) arrow (listof number) throws error
;; Purpose: To evaluate the sexprs in the given list
(define (eval-args a-losexpr)
```
- ```
;; Sample expressions for eval-args
(define LOSEXPRESS1-VAL '())
(define LOSEXPRESS2-VAL (cons (eval-sexpr (first LOSEXPRESS2))
                               (eval-args (rest LOSEXPRESS2))))
```
- ```
;; Tests using sample computations for eval-args
(check-expect (eval-args LOSEXPRESS1) LOSEXPRESS1-VAL)
(check-expect (eval-args LOSEXPRESS2) LOSEXPRESS2-VAL)
```
- ```
;; Tests using sample values for eval-args
(check-expect (eval-args '((- 1 1) 89)) '(0 89))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; (listof sexpr) arrow (listof number) throws error
;; Purpose: To evaluate the sexprs in the given list
(define (eval-args a-losexpr)
```
- ```
(if (empty? a-losexpr)
      '()
      (cons (eval-seexpr (first a-losexpr))
            (eval-args (rest a-losexpr)))))
```
- ```
; Sample expressions for eval-args
(define LOSEXPRESS1-VAL '())
(define LOSEXPRESS2-VAL (cons (eval-seexpr (first LOSEXPRESS2))
 (eval-args (rest LOSEXPRESS2)))))
```
- ```
; Tests using sample computations for eval-args
(check-expect (eval-args LOSEXPRESS1) LOSEXPRESS1-VAL)
(check-expect (eval-args LOSEXPRESS2) LOSEXPRESS2-VAL)

;; Tests using sample values for eval-args
(check-expect (eval-args '((- 1 1) 89)) '(0 89))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ;; Sample expressions for apply-f
 - (define APPLY1-VAL (sum-lon LON1))
 - (define APPLY2-VAL (subt-lon LON2))
 - (define APPLY3-VAL (mult-lon LON3))

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; function (listof number) → number throws error
;; Purpose: Apply the given function to the given numbers
(define (apply-f a-function a-lon)
```
- ```
;; Sample expressions for apply-f
(define APPLY1-VAL (sum-lon LON1))
(define APPLY2-VAL (subt-lon LON2))
(define APPLY3-VAL (mult-lon LON3))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; function (listof number) → number throws error
;; Purpose: Apply the given function to the given numbers
(define (apply-f a-function a-lon)
```
- ```
;; Sample expressions for apply-f
(define APPLY1-VAL (sum-lon LON1))
(define APPLY2-VAL (subt-lon LON2))
(define APPLY3-VAL (mult-lon LON3))
```
- ```
;; Tests using sample computations for apply-f
(check-expect (apply-f '+ LON1) APPLY1-VAL)
(check-expect (apply-f '- LON2) APPLY2-VAL)
(check-expect (apply-f '* LON3) APPLY3-VAL)
```
- ```
; Tests using sample values for apply-f
(check-expect (apply-f '+ '(10 10 10)) 30)
(check-expect (apply-f '- '(20 0)) 20)
(check-expect (apply-f '* '(9 -2)) -18)
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; function (listof number) → number throws error
;; Purpose: Apply the given function to the given numbers
(define (apply-f a-function a-lon))
```
- ```
(cond [(eq? a-function '+) (sum-lon a-lon)] sum-loq in Sect. 72
        [(eq? a-function '-) (subt-lon a-lon)]
        [else (mult-lon a-lon)]))
```
- ```
; Sample expressions for apply-f
(define APPLY1-VAL (sum-lon LON1))
(define APPLY2-VAL (subt-lon LON2))
(define APPLY3-VAL (mult-lon LON3))
```
- ```
; Tests using sample computations for apply-f
(check-expect (apply-f '+ LON1) APPLY1-VAL)
(check-expect (apply-f '- LON2) APPLY2-VAL)
(check-expect (apply-f '* LON3) APPLY3-VAL)
```



```
; Tests using sample values for apply-f
(check-expect (apply-f '+ '(10 10 10)) 30)
(check-expect (apply-f '- '(20 0)) 20)
(check-expect (apply-f '* '(9 -2)) -18)
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; Sample instances of (listof number)
(define LON1 '())
(define LON2 '(1 2 3 4 5))
(define LON3 '(-10 -20 -30))
```

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; Sample instances of (listof number)
(define LON1 '())
(define LON2 '(1 2 3 4 5))
(define LON3 '(-10 -20 -30))
```
- ```
;; Sample expressions for mult-lon
(define LON1-MULT 1)
(define LON2-MULT (* (first LON2) (mult-lon (rest LON2))))
```

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; Sample instances of (listof number)
(define LON1 '())
(define LON2 '(1 2 3 4 5))
(define LON3 '(-10 -20 -30))
```
- ```
;; lon → number
;; Purpose: To multiply the numbers in the given lon
(define (mult-lon a-lon))
```
- ```
;; Sample expressions for mult-lon
(define LON1-MULT 1)
(define LON2-MULT (* (first LON2) (mult-lon (rest LON2))))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; Sample instances of (listof number)
(define LON1 '())
(define LON2 '(1 2 3 4 5))
(define LON3 '(-10 -20 -30))
```
- ```
;; lon → number
;; Purpose: To multiply the numbers in the given lon
(define (mult-lon a-lon))
```
- ```
;; Sample expressions for mult-lon
(define LON1-MULT 1)
(define LON2-MULT (* (first LON2) (mult-lon (rest LON2))))
```
- ```
;; Tests using sample computations for mult-lon
(check-expect (mult-lon LON1) LON1-MULT)
(check-expect (mult-lon LON2) LON2-MULT)
```
- ```
;; Tests using sample computations for mult-lon
(check-expect (mult-lon '(-10 0 10)) 0)
```

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ;; Sample instances of (listof number)  
`(define LON1 '())  
(define LON2 '(1 2 3 4 5))  
(define LON3 '(-10 -20 -30))`
- ;; lon → number  
      ;; Purpose: To multiply the numbers in the given lon  
`(define (mult-lon a-lon)`
- `(if (empty? a-lon)  
          1  
          (* (first a-lon) (mult-lon (rest a-lon))))`
- ;; Sample expressions for mult-lon  
`(define LON1-MULT 1)  
(define LON2-MULT (* (first LON2) (mult-lon (rest LON2))))`
- ;; Tests using sample computations for mult-lon  
`(check-expect (mult-lon LON1) LON1-MULT)  
(check-expect (mult-lon LON2) LON2-MULT)`
- ;; Tests using sample computations for mult-lon  
`(check-expect (mult-lon '(-10 0 10)) 0)`

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments
  - $> (- 1)$   
-1
  - $> (- 2 -4)$   
6
  - $> (- 10 20 -5)$   
-5
- How is `-` applied to its arguments?

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments
  - $> (- 1)$   
-1
  - $> (- 2 -4)$   
6
  - $> (- 10 20 -5)$   
-5
- How is `-` applied to its arguments?
- According the ISL+ documentation this is how subtraction works:  
Subtracts the second (and following) number(s) from the first;  
negates the number if there is only one argument.

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments
  - $> (- 1)$   
-1  
 $> (- 2 -4)$   
6  
 $> (- 10 20 -5)$   
-5
  - How is `-` applied to its arguments?
  - According the ISL+ documentation this is how subtraction works:  
Subtracts the second (and following) number(s) from the first;  
negates the number if there is only one argument.
  - Therefore, we have:

$$(- 1) = -1$$

$$(- 2 -4) = 2 - -4$$

$$(- 10 20 -5) = 10 - 20 - -5$$

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments
  - $> (- 1)$   
 $-1$   
 $> (- 2 -4)$   
 $6$   
 $> (- 10 20 -5)$   
 $-5$
  - How is `-` applied to its arguments?
  - According the ISL+ documentation this is how subtraction works:  
Subtracts the second (and following) number(s) from the first;  
negates the number if there is only one argument.
  - Therefore, we have:

|                |                  |
|----------------|------------------|
| $(- 1)$        | $= -1$           |
| $(- 2 -4)$     | $= 2 - -4$       |
| $(- 10 20 -5)$ | $= 10 - 20 - -5$ |
  - What happens if `-` is provided no arguments?

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments
  - $\gt (- 1)$   
 $-1$   
 $\gt (- 2 -4)$   
 $6$   
 $\gt (- 10 20 -5)$   
 $-5$
  - How is `-` applied to its arguments?
  - According the ISL+ documentation this is how subtraction works:  
Subtracts the second (and following) number(s) from the first;  
negates the number if there is only one argument.
  - Therefore, we have:

|                |                  |
|----------------|------------------|
| $(- 1)$        | $= -1$           |
| $(- 2 -4)$     | $= 2 - -4$       |
| $(- 10 20 -5)$ | $= 10 - 20 - -5$ |
  - What happens if `-` is provided no arguments?
  - An error is thrown

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Of the three auxiliary functions the most interesting one is `subt-lon`
- Analyze how minus is applied to an arbitrary number of arguments
  - $\gt (- 1)$   
 $-1$   
 $\gt (- 2 -4)$   
 $6$   
 $\gt (- 10 20 -5)$   
 $-5$
  - How is `-` applied to its arguments?
  - According the ISL+ documentation this is how subtraction works:  
Subtracts the second (and following) number(s) from the first;  
negates the number if there is only one argument.
  - Therefore, we have:

|                |                  |
|----------------|------------------|
| $(- 1)$        | $= -1$           |
| $(- 2 -4)$     | $= 2 - -4$       |
| $(- 10 20 -5)$ | $= 10 - 20 - -5$ |
  - What happens if `-` is provided no arguments?
  - An error is thrown
  - This is why `eval-expr`, `eval-slist`, `eval-args`, and `apply-f` may throw an error

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Now that we understand how to apply – we can determine how to implement a function to perform this task

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Now that we understand how to apply – we can determine how to implement a function to perform this task
- If the given list of numbers is empty then an error is thrown

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- Now that we understand how to apply – we can determine how to implement a function to perform this task
- If the given list of numbers is empty then an error is thrown
- Otherwise, we need to subtract the rest of the numbers from the first number
- How can this be done?

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Now that we understand how to apply - we can determine how to implement a function to perform this task
- If the given list of numbers is empty then an error is thrown
- Otherwise, we need to subtract the rest of the numbers from the first number
- How can this be done?
- Observe that a - is placed before each number in the rest of the list
- Using a little Algebra we can factor out the -:

$$(- 2 -4) = 2 - -4 = 2 - (-4)$$

$$(- 10 20 -5) = 10 - 20 - -5 = 10 - (20 + -5)$$

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Now that we understand how to apply - we can determine how to implement a function to perform this task
- If the given list of numbers is empty then an error is thrown
- Otherwise, we need to subtract the rest of the numbers from the first number
- How can this be done?
- Observe that a - is placed before each number in the rest of the list
- Using a little Algebra we can factor out the -:  
$$(- 2 -4) = 2 - -4 = 2 - (-4)$$
$$(- 10 20 -5) = 10 - 20 - -5 = 10 - (20 + -5)$$
- Implementation: Subtract from the first number the sum of the rest of the numbers

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Mutually Recursive Data

## Evaluating Arithmetic Expressions

- ```
;; Sample expressions for subt-lon
(define LON2-SUBT (- (first LON2) (sum-lon (rest LON2))))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; lon → number throws error
;; Purpose: To subtract the given lon
(define (subt-lon a-lon))
```
- ```
;; Sample expressions for subt-lon
(define LON2-SUBT (- (first LON2) (sum-lon (rest LON2))))
```

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ;; lon → number throws error
;; Purpose: To subtract the given lon
(define (subt-lon a-lon))
 - ;; Sample expressions for subt-lon
(define LON2-SUBT (- (first LON2) (sum-lon (rest LON2))))
 - ;; Tests using sample computations for subt-lon
(check-error (subt-lon LON1) "No numbers provided to -.")
(check-expect (subt-lon LON2) LON2-SUBT)
- ;; Tests using sample computations for subt-lon
(check-expect (subt-lon '(-10 10 10)) -30)

Mutually Recursive Data

Evaluating Arithmetic Expressions

- ```
;; lon → number throws error
;; Purpose: To subtract the given lon
(define (subt-lon a-lon)
```
- ```
(if (empty? a-lon)
      (error "No numbers provided to -.")
      (- (first a-lon) (sum-lon (rest a-lon)))))
```
- ```
;; Sample expressions for subt-lon
(define LON2-SUBT (- (first LON2) (sum-lon (rest LON2))))
```
- ```
;; Tests using sample computations for subt-lon
(check-error (subt-lon LON1) "No numbers provided to -.")
(check-expect (subt-lon LON2) LON2-SUBT)

;; Tests using sample computations for subt-lon
(check-expect (subt-lon '(-10 10 10)) -30)
```

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

- The concept of a binary tree may be generalized
- Instead of each node having at most two children, in a *tree* a node may have an arbitrary number of children

Part III: Compound Data of Arbitrary Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

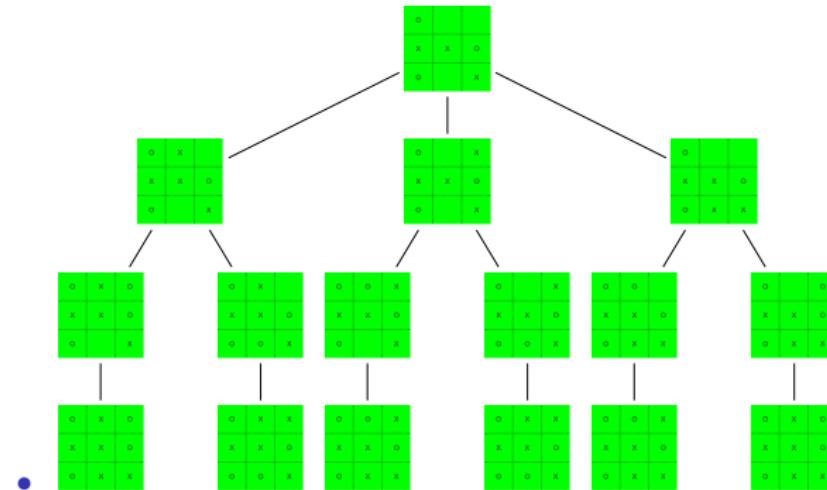
Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

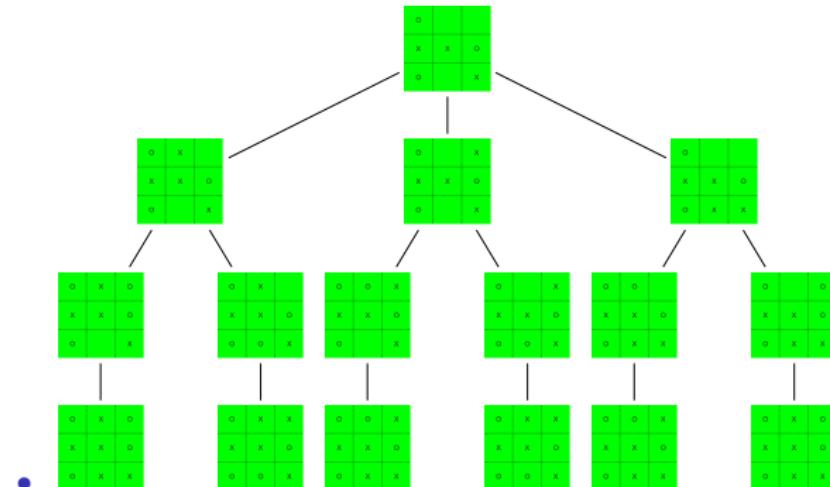
Processing
Multiple
Inputs of
Arbitrary
Size

Trees



- The concept of a binary tree may be generalized
- Instead of each node having at most two children, in a *tree* a node may have an arbitrary number of children

Trees



- The concept of a binary tree may be generalized
- Instead of each node having at most two children, in a *tree* a node may have an arbitrary number of children
- Trees are versatile and may be used to represent many real or imaginary objects
- One use of trees is to represent a *search space*
- A search space defines all possible solutions to a problem and is searched to find a solution
- Tree above represents all paths Tic Tac Toe game starting at the root node
- May be function input to determine the next move

**Part III:
Compound
Data of
Arbitrary
Size**

**Marco T.
Morazán**

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

- Must now decide how to represent a search tree for Tic Tac Toe

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

- Must now decide how to represent a search tree for Tic Tac Toe
- Observe that a search tree is made of nodes
- Suggests that two data definitions are needed: one for a node and one for a search tree

Trees

- Must now decide how to represent a search tree for Tic Tac Toe
- Observe that a search tree is made of nodes
- A node has a board and a list of children

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

- Must now decide how to represent a search tree for Tic Tac Toe
 - Observe that a search tree is made of nodes
-
- A node has a board and a list of children
 - Observe two more things:
 - Need for a board data definition
 - A node always contains two elements

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

- Must now decide how to represent a search tree for Tic Tac Toe
 - Observe that a search tree is made of nodes
-
- A node has a board and a list of children
 - Observe two more things:
 - Need for a board data definition
 - A node always contains two elements
 - Think carefully about what a search tree is

Trees

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- Must now decide how to represent a search tree for Tic Tac Toe
 - Observe that a search tree is made of nodes
-
- A node has a board and a list of children
 - Observe two more things:
 - Need for a board data definition
 - A node always contains two elements
 - Think carefully about what a search tree is
 - It is '()' when there is not game
 - If there is a game then the search tree must be a node

Trees

- ```
;; A node is a structure: (make-node board (listof st))
(define-struct node (board children))

;; A solution tree (st) is either:
;; 1. '()
;; 2. node

;; A board value (bval) is either:
;; 1. 'X 2. 'O 3. 'B
;; A board is a structure:
;; (make-board bval bval bval bval bval bval bval bval)
(define-struct board (p0 p1 p2 p3 p4 p5 p6 p7 p8))

;; Sample board values
(define INIT-BOARD (make-board 'B 'B 'B
 'B 'B 'B
 'B 'B 'B))
(define BOARD1 (make-board 'X 'O 'X
 'O 'X 'O
 'X 'O 'X))
(define BOARD2 (make-board 'X 'B 'B
 'O 'B 'B
 'B 'B 'B))
(define BOARD3 (make-board 'X 'B 'B
 'X 'O 'B
 'B 'B 'B))
(define BOARD4 (make-board 'X 'B 'B
 'B 'B 'B
 'B 'B 'B))
(define BOARD5 (make-board 'X 'X 'O
 'B 'X 'X
 'O 'O 'O))
(define BOARD6 (make-board 'X 'B 'B
 'B 'X 'X
 'O 'B 'O))
```

## Part III: Compound Data of Arbitrary Size

Marco T.  
Morazán

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Trees

- #| TEMPLATES  
;; Sample instances of (listof st)  
(define LOST1 '()) (define LOST2 ...) ...  
  
;; (listof st) ... → ... Purpose:  
(define (f-on-lost an-lost)  
 (if (empty? an-lost)  
 ...  
 ...  
 ...(f-on-st (first an-lost))...(f-on-lost (rest an-lost))))  
  
;; Sample expressions for f-on-lost  
(define ST1-VAL ...)  
(define ST2-VAL ...)  
  
;; Tests using sample computations f-on-lost  
(check-expect (f-on-lost ST1 ...) ST1-VAL)  
(check-expect (f-on-lost ST2 ...) ST2-VAL)  
;; Tests using sample values f-on-lost  
(check-expect (f-on-lost .....) ....)

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Trees

- ```
;; Sample instances of node
(define NODE1 ...)  ...

;; node ... → ... Purpose:
(define (f-on-node a-node)
  ... (f-on-board (node-board a-node))...
  ... (f-on-lost  (node-children a-node))...)

;; Sample expressions for f-on-node
(define NODE1-VAL ...)

;; Tests using sample computations f-on-node
(check-expect (f-on-node NODE1 ...) NODE1-VAL)
;; Tests using sample values f-on-node
(check-expect (f-on-node ..... ) ...)
```

Trees

- ```
;; Sample instances of ST
(define ST1 '()) (define ST2 ...) ...

;; st ... → ... Purpose:
(define (f-on-st an-st)
 (if (empty? an-st)
 ...
 (f-on-node an-st)))

;; Sample expressions for f-on-st
(define ST1-VAL ...)
(define ST2-VAL ...)

;; Tests using sample computations f-on-st
(check-expect (f-on-node ST1 ...) ST1-VAL)
(check-expect (f-on-node ST2 ...) ST2-VAL)
;; Tests using sample values f-on-st
(check-expect (f-on-st) ...) | #
```

# Trees

## Creating an Search Tree for Tic Tac Toe

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- How do we create a tree representing the complete st for Tic Tac Toe starting from the blank board?

# Trees

## Creating an Search Tree for Tic Tac Toe

- How do we create a tree representing the complete st for Tic Tac Toe starting from the blank board?
- To create an st we need to know whose turn it is
- Create a data definition and sample instances:

```
;; A turn is either 'X or 'O
```

```
(define INIT-TURN 'X)
(define O-TURN 'O)
```

# Trees

## Creating an Search Tree for Tic Tac Toe

- How do we create a tree representing the complete st for Tic Tac Toe starting from the blank board?
- To create an st we need to know whose turn it is
- Create a data definition and sample instances:

```
;; A turn is either 'X or 'O
```

```
(define INIT-TURN 'X)
(define O-TURN 'O)
```

- Given a board and a turn the blanks in the board must be identified
- A child for the given board must be created for each blank.

# Trees

## Creating an Search Tree for Tic Tac Toe

- How do we create a tree representing the complete st for Tic Tac Toe starting from the blank board?
- To create an st we need to know whose turn it is
- Create a data definition and sample instances:

```
;; A turn is either 'X or 'O
```

```
(define INIT-TURN 'X)
(define O-TURN 'O)
```

- Given a board and a turn the blanks in the board must be identified
- A child for the given board must be created for each blank.
- A node is created using the given board and the sts created by processing the blanks of the given board

Part III:  
Compound  
Data of  
Arbitrary  
Size

Marco T.  
Morazán

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Trees

## Creating an Search Tree for Tic Tac Toe

- ```
;; Sample expressions for create-st
(define BOARD1-VAL (make-node BOARD1
    (process-blanks
        BOARD1
        (find-blanks BOARD1)
        '0)))
(define BOARD2-VAL (make-node BOARD2
    (process-blanks
        BOARD2
        (find-blanks BOARD2)
        'X)))
```

Trees

Creating an Search Tree for Tic Tac Toe

- ```
;; board turn → st
;; Purpose: Create the st for the given board
(define (create-st a-board a-turn)
```
- ```
; Sample expressions for create-st
(define BOARD1-VAL (make-node BOARD1
  (process-blanks
    BOARD1
    (find-blanks BOARD1)
    'O)))
(define BOARD2-VAL (make-node BOARD2
  (process-blanks
    BOARD2
    (find-blanks BOARD2)
    'X)))
```

Trees

Creating an Search Tree for Tic Tac Toe

- ```
;; board turn → st
;; Purpose: Create the st for the given board
(define (create-st a-board a-turn)
```
- ```
; Sample expressions for create-st
(define BOARD1-VAL (make-node BOARD1
  (process-blanks
   BOARD1
   (find-blanks BOARD1)
   'O)))
(define BOARD2-VAL (make-node BOARD2
  (process-blanks
   BOARD2
   (find-blanks BOARD2)
   'X)))
```
- ```
; Tests using sample computations for create-st
(check-expect (create-st BOARD1 'O) BOARD1-VAL)
(check-expect (create-st BOARD2 'X) BOARD2-VAL)
```

# Trees

## Creating an Search Tree for Tic Tac Toe

- ```
;; board turn → st
;; Purpose: Create the st for the given board
(define (create-st a-board a-turn)
  (make-node a-board
    (process-blanks a-board
      (find-blanks a-board)
      a-turn)))
```
- ```
;; Sample expressions for create-st
(define BOARD1-VAL (make-node BOARD1
 (process-blanks
 BOARD1
 (find-blanks BOARD1)
 'O)))
(define BOARD2-VAL (make-node BOARD2
 (process-blanks
 BOARD2
 (find-blanks BOARD2)
 'X)))
```
- ```
;; Tests using sample computations for create-st
(check-expect (create-st BOARD1 'O) BOARD1-VAL)
(check-expect (create-st BOARD2 'X) BOARD2-VAL)
```

Trees

Creating an Search Tree for Tic Tac Toe

- ;; Tests using sample values for create-st
(check-expect (create-st (make-board 'X 'O 'X
'O 'X 'O
'X 'B 'B)
'O)
(make-node
(make-board 'X 'O 'X
'O 'X 'O
'X 'B 'B)
(list
(make-node
(make-board 'X 'O 'X
'O 'X 'O
'X 'O 'B)
(list
(make-node
(make-board 'X 'O 'X
'O 'X 'O
'X 'O 'X)
'()))))
(make-node
(make-board 'X 'O 'X
'O 'X 'O
'X 'B 'O)
(list
(make-node
(make-board 'X 'O 'X
'O 'X 'O
'X 'X 'O)
'()))))))

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

Creating an Search Tree for Tic Tac Toe

- We can now turn our attention to the design of process-blanks

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

Creating an Search Tree for Tic Tac Toe

- We can now turn our attention to the design of `process-blanks`
- Must process a (`listof bpos`)—list of board positions in [0..8]

Trees

Creating an Search Tree for Tic Tac Toe

- We can now turn our attention to the design of `process-blanks`
- Must process a (`listof bpos`)—list of board positions in `[0..8]`
- For each blank position in the given list a new board is created by placing the given `bval` at that position in the given board
- This board is then used to create an `st` for the (`listof st`) returned

Trees

Creating an Search Tree for Tic Tac Toe

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ;; Sample expressions for process-blanks
(define PROC-0BLANKS '())
(define PROC-6BLANKS
 (cons (create-st (place-on-board
 BOARD3 (first '(1 2 5 6 7 8)) '0)
 (if (eq? '0 'X) '0 'X))
 (process-blanks
 BOARD3 (rest '(1 2 5 6 7 8)) '0))))

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

Creating an Search Tree for Tic Tac Toe

- ```
;; board (listof bpos) turn → (listof st)
;; Purpose: Build the st for board and list of blank positions
(define (process-blanks a-board a-lbpos a-turn)
```

- ```
;; Sample expressions for process-blanks
(define PROC-0BLANKS '())
(define PROC-6BLANKS
  (cons (create-st (place-on-board
                     BOARD3 (first '(1 2 5 6 7 8)) '0)
                  (if (eq? '0 'X) '0 'X))
        (process-blanks
         BOARD3 (rest '(1 2 5 6 7 8)) '0))))
```

Trees

Creating an Search Tree for Tic Tac Toe

- ;; board (listof bpos) turn → (listof st)
;; Purpose: Build the st for board and list of blank positions
(define (process-blanks a-board a-lbpos a-turn)

- ;; Sample expressions for process-blanks
(define PROC-OBLANKS '())
(define PROC-6BLANKS
 (cons (create-st (place-on-board
 BOARD3 (first '(1 2 5 6 7 8)) '0)
 (if (eq? '0 'X) '0 'X))
 (process-blanks
 BOARD3 (rest '(1 2 5 6 7 8)) '0)))
- ;; Tests using sample computations for process-blanks
(check-expect (process-blanks BOARD1 '())
 PROC-OBLANKS)
(check-expect (process-blanks BOARD3 '(1 2 5 6 7 8) '0)
 PROC-6BLANKS)
;; Tests using sample values for process-blanks
(check-expect (process-blanks
 (make-board 'B 'X '0 '0 'X 'X '0 'X '0)
 '(0)
 'X)
 (list (make-node
 (make-board 'X 'X '0 '0 'X 'X '0 'X '0)
 '()))))

Trees

Creating an Search Tree for Tic Tac Toe

- ```
;; board (listof bpos) turn → (listof st)
;; Purpose: Build the st for board and list of blank positions
(define (process-blanks a-board a-lbpos a-turn)

 • (if (empty? a-lbpos)
 '()
 (cons (create-st
 (place-on-board a-board (first a-lbpos) a-turn)
 (if (eq? a-turn 'X) 'O 'X))
 (process-blanks a-board (rest a-lbpos) a-turn))))
```
- ```
; Sample expressions for process-blanks
(define PROC-0BLANKS '())
(define PROC-6BLANKS
  (cons (create-st (place-on-board
                     BOARD3 (first '(1 2 5 6 7 8)) 'O)
                     (if (eq? 'O 'X) 'O 'X))
        (process-blanks
         BOARD3 (rest '(1 2 5 6 7 8)) 'O)))
```
- ```
; Tests using sample computations for process-blanks
(check-expect (process-blanks BOARD1 '() 'X)
 PROC-0BLANKS)
(check-expect (process-blanks BOARD3 '(1 2 5 6 7 8) 'O)
 PROC-6BLANKS)
; Tests using sample values for process-blanks
(check-expect (process-blanks
 (make-board 'B 'X 'O 'O 'X 'X 'O 'X 'O)
 '()
 'X)
 (list (make-node
 (make-board 'X 'X 'O 'O 'X 'X 'O 'X 'O)
 '()))))
```

**Part III:  
Compound  
Data of  
Arbitrary  
Size**

**Marco T.  
Morazán**

**Lists**

**List  
Processing**

**Natural  
Numbers**

**Interval  
Processing**

**Aliens Attack  
Version 4**

**Binary Trees**

**Mutually  
Recursive  
Data**

**Processing  
Multiple  
Inputs of  
Arbitrary  
Size**

# Trees

Can Win Tic Tac Toe?

- We know turn to an example of processing an st
- Given an st consider determining if it is possible for a given player to win by any number of moves

# Trees

## Can Win Tic Tac Toe?

- We know turn to an example of processing an st
- Given an st consider determining if it is possible for a given player to win by any number of moves
- The root board and all the boards under it must be checked for a win by the given turn
- If the given st is empty then the answer is #false
- Otherwise, according to the template for a function on an st, a function to determine if the given turn may win in any number of moves for a given node must be called

Part III:  
Compound  
Data of  
Arbitrary  
Size

Marco T.  
Morazán

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Trees

Can Win Tic Tac Toe?

- ```
;; Sample st instances
(define INIT-ST (create-st INIT-BOARD 'X))
(define BRD7-ST (create-st BOARD7      '0))
```

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

Can Win Tic Tac Toe?

- ```
;; Sample st instances
(define INIT-ST (create-st INIT-BOARD 'X))
(define BRD7-ST (create-st BOARD7 '0))
```

- ```
;; Sample expressions for st-can-win?
(define WIN-ST1-VAL #false)
(define WIN-ST2-VAL (node-can-win? INIT-ST 'X))
(define WIN-ST3-VAL (node-can-win? BRD7-ST '0))
```

Trees

Can Win Tic Tac Toe?

- ```
;; Sample st instances
(define INIT-ST (create-st INIT-BOARD 'X))
(define BRD7-ST (create-st BOARD7 '0))
```
- ```
;; st turn → Boolean
;; Purpose: Determine if there is a sequence of moves for
;;           the given turn to win
(define (st-can-win? an-st a-turn)
```
- ```
;; Sample expressions for st-can-win?
(define WIN-ST1-VAL #false)
(define WIN-ST2-VAL (node-can-win? INIT-ST 'X))
(define WIN-ST3-VAL (node-can-win? BRD7-ST '0))
```

# Trees

## Can Win Tic Tac Toe?

- ```
;; Sample st instances
(define INIT-ST (create-st INIT-BOARD 'X))
(define BRD7-ST (create-st BOARD7      '0))
```
- ```
;; st turn → Boolean
;; Purpose: Determine if there is a sequence of moves for
;; the given turn to win
(define (st-can-win? an-st a-turn)
```

- ```
;; Sample expressions for st-can-win?
(define WIN-ST1-VAL #false)
(define WIN-ST2-VAL (node-can-win? INIT-ST 'X))
(define WIN-ST3-VAL (node-can-win? BRD7-ST '0))
```
- ```
; Tests using sample computations st-can-win?
(check-expect (st-can-win? WIN-ST1 'X) WIN-ST1-VAL)
(check-expect (st-can-win? WIN-ST2 'X) WIN-ST2-VAL)
(check-expect (st-can-win? WIN-ST3 '0) WIN-ST3-VAL)
```

```
; Tests using sample values can-win-st?
(check-expect (st-can-win? (create-st (make-board '0 'X 'B
 'X '0 'X
 '0 'X '0)
 'X)
 #false)

 (check-expect (st-can-win? (create-st (make-board 'B 'B 'B
 'X '0 'X
 'B 'B 'B)
 '0)
 '0))
```

# Trees

## Can Win Tic Tac Toe?

- ```
;; Sample st instances
(define INIT-ST (create-st INIT-BOARD 'X))
(define BRD7-ST (create-st BOARD7      'O))
```
- ```
;; st turn → Boolean
;; Purpose: Determine if there is a sequence of moves for
;; the given turn to win
(define (st-can-win? an-st a-turn)
 (if (empty? an-st)
 #false
 (node-can-win? an-st a-turn)))
```
- ```
; Sample expressions for st-can-win?
(define WIN-ST1-VAL #false)
(define WIN-ST2-VAL (node-can-win? INIT-ST 'X))
(define WIN-ST3-VAL (node-can-win? BRD7-ST 'O))
```
- ```
; Tests using sample computations st-can-win?
(check-expect (st-can-win? WIN-ST1 'X) WIN-ST1-VAL)
(check-expect (st-can-win? WIN-ST2 'X) WIN-ST2-VAL)
(check-expect (st-can-win? WIN-ST3 'O) WIN-ST3-VAL)
```

;; Tests using sample values can-win-st?

```
(check-expect (st-can-win? (create-st (make-board 'O 'X 'B
 'X 'O 'X
 'O 'X 'O)
 'X)
 #false))

 (check-expect (st-can-win? (create-st (make-board 'B 'B 'B
 'X 'O 'X
 'B 'B 'B)
 'O)
 #false))
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Trees

## Can Win Tic Tac Toe?

- Now turn our attention to designing node-can-win?
- Recall that a node has a board and a list of children sts
- What does it mean that the given turn can win?

# Trees

## Can Win Tic Tac Toe?

- Now turn our attention to designing `node-can-win`?
- Recall that a node has a board and a list of children sts
- What does it mean that the given turn can win?
- The given turn can win if it has won in the node's board
- This means two things
  - The given turn has three of its values in any row
  - The given turn's opponent has not won
- The given turn can also win if it can win through any of the given node's children

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Trees

Can Win Tic Tac Toe?

- ;; Sample instances of node  
(define WIN-NODE2 WIN-ST2)  
(define WIN-NODE3 WIN-ST3)

# Trees

## Can Win Tic Tac Toe?

- ;; Sample instances of node  

```
(define WIN-NODE2 WIN-ST2)
(define WIN-NODE3 WIN-ST3)
```
- ;; Sample expressions for node-can-win?  

```
(define WIN-NODE2-VAL
 (or
 (and (not (has-win? (node-board WIN-NODE2) (flip 'X)))
 (has-win? (node-board WIN-NODE2) 'X))
 (lost-can-win? (node-children WIN-NODE2) 'X)))
 (define WIN-NODE3-VAL
 (or (and (not (has-win? (node-board WIN-NODE3) (flip 'O)))
 (has-win? (node-board WIN-NODE3) 'O))
 (lost-can-win? (node-children WIN-NODE3) 'O)))
```

# Trees

## Can Win Tic Tac Toe?

- ;; Sample instances of node  
(define WIN-NODE2 WIN-ST2)  
(define WIN-NODE3 WIN-ST3)
- ;; node turn → Boolean Purpose: Determine if the given turn can win  
(define (node-can-win? a-node a-turn)
- ;; Sample expressions for node-can-win?  
(define WIN-NODE2-VAL  
 (or  
 (and (not (has-win? (node-board WIN-NODE2) (flip 'X)))  
 (has-win? (node-board WIN-NODE2) 'X))  
 (lost-can-win? (node-children WIN-NODE2) 'X)))  
  
(define WIN-NODE3-VAL  
 (or (and (not (has-win? (node-board WIN-NODE3) (flip '0)))  
 (has-win? (node-board WIN-NODE3) '0))  
 (lost-can-win? (node-children WIN-NODE3) '0)))

# Trees

## Can Win Tic Tac Toe?

- ```
;; Sample instances of node
(define WIN-NODE2 WIN-ST2)
(define WIN-NODE3 WIN-ST3)
```
- ```
; node turn → Boolean Purpose: Determine if the given turn can win
(define (node-can-win? a-node a-turn)
```
- ```
; Sample expressions for node-can-win?
(define WIN-NODE2-VAL
  (or
    (and (not (has-win? (node-board WIN-NODE2) (flip 'X)))
          (has-win? (node-board WIN-NODE2) 'X))
         (lost-can-win? (node-children WIN-NODE2) 'X)))

(define WIN-NODE3-VAL
  (or (and (not (has-win? (node-board WIN-NODE3) (flip 'O)))
            (has-win? (node-board WIN-NODE3) 'O))
       (lost-can-win? (node-children WIN-NODE3) 'O)))
```
- ```
; Tests using sample computations node-can-win?
(check-expect (node-can-win? WIN-NODE2 'X) WIN-NODE2-VAL)
(check-expect (node-can-win? WIN-NODE3 'O) WIN-NODE3-VAL)

;; Tests using sample values node-can-win?
(check-expect
 (node-can-win? (create-st (make-board 'B 'B 'B 'B 'X 'B 'B 'B) 'O) 'O)
 #true)
(check-expect
 (node-can-win? (create-st (make-board 'O 'O 'O 'X 'X 'B 'X 'B) 'X) 'X)
 #false)
```

# Trees

## Can Win Tic Tac Toe?

- ```
;; Sample instances of node
(define WIN-NODE2 WIN-ST2)
(define WIN-NODE3 WIN-ST3)
```
- ```
; node turn → Boolean Purpose: Determine if the given turn can win
(define (node-can-win? a-node a-turn)
 (or (and (not (has-win? (node-board a-node) (flip a-turn)))
 (has-win? (node-board a-node) a-turn))
 (lost-can-win? (node-children a-node) a-turn)))
```
- ```
; Sample expressions for node-can-win?
(define WIN-NODE2-VAL
  (or
    (and (not (has-win? (node-board WIN-NODE2) (flip 'X)))
          (has-win? (node-board WIN-NODE2) 'X))
        (lost-can-win? (node-children WIN-NODE2) 'X)))
```
- ```
(define WIN-NODE3-VAL
 (or (and (not (has-win? (node-board WIN-NODE3) (flip '0)))
 (has-win? (node-board WIN-NODE3) '0))
 (lost-can-win? (node-children WIN-NODE3) '0)))
```
- ```
; Tests using sample computations node-can-win?
(check-expect (node-can-win? WIN-NODE2 'X) WIN-NODE2-VAL)
(check-expect (node-can-win? WIN-NODE3 '0) WIN-NODE3-VAL)

;; Tests using sample values node-can-win?
(check-expect
  (node-can-win? (create-st (make-board 'B 'B 'B 'B 'X 'B 'B 'B 'B) '0) '0)
  #true)
(check-expect
  (node-can-win? (create-st (make-board '0 '0 '0 'X 'X 'B 'X 'B 'B) 'X) 'X)
  #false)
```

Part III:
Compound
Data of
Arbitrary
Size

Marco T.
Morazán

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

Can Win Tic Tac Toe?

- `lost-can-win?` must determine if for a given turn any of its sts lead to a win

Trees

Can Win Tic Tac Toe?

- `lost-can-win?` must determine if for a given turn any of its sts lead to a win
- If the given list is empty then the answer is `#false`
- Otherwise, a win is possible if the first st leads to a win or any of the rest of the sts leads to a win
- Observe that a Boolean value is computed
- This means we may opt to use a Boolean function without using a conditional expression as suggested by the template for a function on a lost

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Trees

Can Win Tic Tac Toe?

- ```
;; Sample instances of lost
(define WIN-LOST1 '())
(define WIN-LOST2 (node-children WIN-NODE2))
(define WIN-LOST3 (node-children WIN-NODE3))
```

# Trees

## Can Win Tic Tac Toe?

- ```
;; Sample instances of lost
(define WIN-LOST1 '())    (define WIN-LOST2 (node-children WIN-NODE2))
(define WIN-LOST3 (node-children WIN-NODE3))
```

- ```
;; Sample expressions for lost-can-win?
(define WIN-LOST1-VAL (and (not (empty? WIN-LOST1))
 (or (st-can-win? (first WIN-LOST1) 'X)
 (lost-can-win? (rest WIN-LOST1) 'X))))
(define WIN-LOST2-VAL (and (not (empty? WIN-LOST2))
 (or (st-can-win? (first WIN-LOST2) 'X)
 (lost-can-win? (rest WIN-LOST2) 'X))))
(define WIN-LOST3-VAL (and (not (empty? WIN-LOST3))
 (or (st-can-win? (first WIN-LOST3) '0)
 (lost-can-win? (rest WIN-LOST3) '0))))
```

# Trees

## Can Win Tic Tac Toe?

- ;; Sample instances of lost  
(define WIN-LOST1 '()) (define WIN-LOST2 (node-children WIN-NODE2))  
(define WIN-LOST3 (node-children WIN-NODE3))
- ;; (listof st) turn → Boolean Purpose: Determine if turn can win  
(define (lost-can-win? an-lost a-turn)
- ;; Sample expressions for lost-can-win?  
(define WIN-LOST1-VAL (and (not (empty? WIN-LOST1))  
                                  (or (st-can-win? (first WIN-LOST1) 'X)  
                                  (lost-can-win? (rest WIN-LOST1) 'X))))  
(define WIN-LOST2-VAL (and (not (empty? WIN-LOST2))  
                                  (or (st-can-win? (first WIN-LOST2) 'X)  
                                  (lost-can-win? (rest WIN-LOST2) 'X))))  
(define WIN-LOST3-VAL (and (not (empty? WIN-LOST3))  
                                  (or (st-can-win? (first WIN-LOST3) '0)  
                                  (lost-can-win? (rest WIN-LOST3) '0))))

# Trees

## Can Win Tic Tac Toe?

- ```
;; Sample instances of lost
(define WIN-LOST1 '())  (define WIN-LOST2 (node-children WIN-NODE2))
(define WIN-LOST3 (node-children WIN-NODE3))
```
- ```
;; (listof st) turn → Boolean Purpose: Determine if turn can win
(define (lost-can-win? an-lost a-turn)
```
- ```
;; Sample expressions for lost-can-win?
(define WIN-LOST1-VAL (and (not (empty? WIN-LOST1))
                           (or (st-can-win? (first WIN-LOST1) 'X)
                               (lost-can-win? (rest WIN-LOST1) 'X))))
(define WIN-LOST2-VAL (and (not (empty? WIN-LOST2))
                           (or (st-can-win? (first WIN-LOST2) 'X)
                               (lost-can-win? (rest WIN-LOST2) 'X))))
(define WIN-LOST3-VAL (and (not (empty? WIN-LOST3))
                           (or (st-can-win? (first WIN-LOST3) '0)
                               (lost-can-win? (rest WIN-LOST3) '0))))
```
- ```
;; Tests using sample computations lost-can-win?
(check-expect (lost-can-win? WIN-LOST1 'X) WIN-LOST1-VAL)
(check-expect (lost-can-win? WIN-LOST2 '0) WIN-LOST2-VAL)
(check-expect (lost-can-win? WIN-LOST3 '0) WIN-LOST3-VAL)
```
- ```
;; Tests using sample values lost-can-win?
(check-expect
  (lost-can-win? (node-children (create-st (make-board 'X 'B '0 '0 'X '0 'B 'X 'X) '0))
                 '0)
  #false)

(check-expect
  (lost-can-win? (node-children (create-st (make-board 'B 'B '0 '0 'X '0 'B 'X 'X) 'X))
                 'X)
  #true)
```

Trees

Can Win Tic Tac Toe?

- ```
;; Sample instances of lost
(define WIN-LOST1 '()) (define WIN-LOST2 (node-children WIN-NODE2))
(define WIN-LOST3 (node-children WIN-NODE3))
```
- ```
; (listof st) turn → Boolean Purpose: Determine if turn can win
(define (lost-can-win? an-lost a-turn)
  (and (not (empty? an-lost)) (or (st-can-win? (first an-lost) a-turn)
                                    (lost-can-win? (rest an-lost) a-turn))))
```
- ```
; Sample expressions for lost-can-win?
(define WIN-LOST1-VAL (and (not (empty? WIN-LOST1))
 (or (st-can-win? (first WIN-LOST1) 'X)
 (lost-can-win? (rest WIN-LOST1) 'X))))
(define WIN-LOST2-VAL (and (not (empty? WIN-LOST2))
 (or (st-can-win? (first WIN-LOST2) 'X)
 (lost-can-win? (rest WIN-LOST2) 'X))))
(define WIN-LOST3-VAL (and (not (empty? WIN-LOST3))
 (or (st-can-win? (first WIN-LOST3) '0)
 (lost-can-win? (rest WIN-LOST3) '0))))
```
- ```
; Tests using sample computations lost-can-win?
(check-expect (lost-can-win? WIN-LOST1 'X) WIN-LOST1-VAL)
(check-expect (lost-can-win? WIN-LOST2 '0) WIN-LOST2-VAL)
(check-expect (lost-can-win? WIN-LOST3 '0) WIN-LOST3-VAL)
```
- ```
; Tests using sample values lost-can-win?
(check-expect
 (lost-can-win? (node-children (create-st (make-board 'X 'B '0 '0 'X '0 'B 'X 'X) '0))
 '0)
 #false)

(check-expect
 (lost-can-win? (node-children (create-st (make-board 'B 'B '0 '0 'X '0 'B 'X 'X) 'X))
 'X)
 #true)
```

# Trees

## Homework

- Problems: 195, 199–200
- Double Bonus Quiz (substitute two lowest quiz grades)  
Problems: 201--212

Due: 1.5 weeks (before class)

Work in groups

# Processing Multiple Inputs of Arbitrary Size

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Our focus up to now has mostly been on the design of functions that consume a single instance of data of arbitrary size
- How are functions that consume multiple instances of data of arbitrary size?

# Processing Multiple Inputs of Arbitrary Size

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Our focus up to now has mostly been on the design of functions that consume a single instance of data of arbitrary size
- How are functions that consume multiple instances of data of arbitrary size?
- To simplify the discussion we shall focus on designing functions that consume two inputs of arbitrary size

# Processing Multiple Inputs of Arbitrary Size

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- Our focus up to now has mostly been on the design of functions that consume a single instance of data of arbitrary size
- How are functions that consume multiple instances of data of arbitrary size?
- To simplify the discussion we shall focus on designing functions that consume two inputs of arbitrary size
- We outline three basic properties that you may establish once problem analysis is performed:
  - One input has the dominant role
  - The two inputs must be processed simultaneously
  - There is no clear relationship between the inputs

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- Imagine that ISL+ did not include a function, `append`, to append two given lists
- If the programming language you are using does not include a function you need then you must design your own
- How you design and implement a function to append two lists?

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- Imagine that ISL+ did not include a function, `append`, to append two given lists
- If the programming language you are using does not include a function you need then you must design your own
- How you design and implement a function to append two lists?
- Always start with problem analysis

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- Imagine that ISL+ did not include a function, `append`, to append two given lists
- If the programming language you are using does not include a function you need then you must design your own
- How you design and implement a function to append two lists?
- Always start with problem analysis
- We are given L1 and L2
- A new list must be created that has all the elements of L1 followed by all the elements of L2

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- Imagine that ISL+ did not include a function, `append`, to append two given lists
- If the programming language you are using does not include a function you need then you must design your own
- How you design and implement a function to append two lists?
- Always start with problem analysis
- We are given L1 and L2
- A new list must be created that has all the elements of L1 followed by all the elements of L2
- To add all the elements of L1 to the resulting list L1 must be traversed
- Do we need to also traverse L2?

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- Imagine that ISL+ did not include a function, `append`, to append two given lists
- If the programming language you are using does not include a function you need then you must design your own
- How you design and implement a function to append two lists?
- Always start with problem analysis
- We are given L1 and L2
- A new list must be created that has all the elements of L1 followed by all the elements of L2
- To add all the elements of L1 to the resulting list L1 must be traversed
- Do we need to also traverse L2?
- Consider what occurs with the last element of L1
- This element must be consed with the result of appending the rest of L1, which is empty, and L2
- What ought to be the result of appending empty with L2?

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- Imagine that ISL+ did not include a function, `append`, to append two given lists
- If the programming language you are using does not include a function you need then you must design your own
- How you design and implement a function to append two lists?
- Always start with problem analysis
- We are given  $L_1$  and  $L_2$
- A new list must be created that has all the elements of  $L_1$  followed by all the elements of  $L_2$
- To add all the elements of  $L_1$  to the resulting list  $L_1$  must be traversed
- Do we need to also traverse  $L_2$ ?
- Consider what occurs with the last element of  $L_1$
- This element must be consed with the result of appending the rest of  $L_1$ , which is empty, and  $L_2$
- What ought to be the result of appending empty with  $L_2$ ?
- $L_2$  is the answer
- Observe that  $L_2$  does not need to be traversed
- $L_1$  plays a dominant role
- This informs us that we ought to design this function around processing  $L_1$

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Processing Multiple Inputs of Arbitrary Size

One Input Has a Dominant Role

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- We shall be careful about data analysis
- The types of the two inputs may be defined as `(listof X)` and `(listof Y)`
- The two inputs may or may not be of the same type
- What is the type of the returned list?

# Processing Multiple Inputs of Arbitrary Size

One Input Has a Dominant Role

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- We shall be careful about data analysis
- The types of the two inputs may be defined as (listof X) and (listof Y)
- The two inputs may or may not be of the same type
- What is the type of the returned list?
- If  $X \neq Y$  then the result is neither of type (listof X) nor (listof Y)

# Processing Multiple Inputs of Arbitrary Size

One Input Has a Dominant Role

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- We shall be careful about data analysis
- The types of the two inputs may be defined as `(listof X)` and `(listof Y)`
- The two inputs may or may not be of the same type
- What is the type of the returned list?
- If  $X \neq Y$  then the result is neither of type `(listof X)` nor `(listof Y)`
- Every element of the result list is either of type X or Y
- There is variety among the elements of the result list

# Processing Multiple Inputs of Arbitrary Size

One Input Has a Dominant Role

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- We shall be careful about data analysis
  - The types of the two inputs may be defined as `(listof X)` and `(listof Y)`
  - The two inputs may or may not be of the same type
  - What is the type of the returned list?
  - If  $X \neq Y$  then the result is neither of type `(listof X)` nor `(listof Y)`
  - Every element of the result list is either of type X or Y
  - There is variety among the elements of the result list
  - Data definition:
    - An XY is either:
      1. X
      2. Y
- We can now write the returned type as `(listof XY)`

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- ```
;; Sample instances of (listof X), where X = number
(define LON1 '())
(define LON2 '(10 20 30 40))

;; Sample instances of (listof Y), where Y = string
(define LOS1 '())
(define LOS2 ('" "A" "BB"))
```

Processing Multiple Inputs of Arbitrary Size

One Input Has a Dominant Role

- ;; Sample instances of (listof X), where X = number
(define LON1 '())
(define LON2 '(10 20 30 40))

```
;; Sample instances of (listof Y), where Y = string  
(define LOS1 '())  
(define LOS2 ('" "A" "BB"))
```

- ;; Sample expressions for myappend
(define LON1-LOS1-VAL LOS1) append LON1 and LOS1
(define LON1-LON2-VAL LON2) append LON1 and LON2
(define LOS2-LOS1-VAL (cons (first LOS2) (myappend (rest LOS2) LOS1)))
(define LON2-LOS2-VAL (cons (first LON2) (myappend (rest LON2) LOS2)))

Processing Multiple Inputs of Arbitrary Size

One Input Has a Dominant Role

- ```
;; Sample instances of (listof X), where X = number
(define LON1 '())
(define LON2 '(10 20 30 40))

;; Sample instances of (listof Y), where Y = string
(define LOS1 '())
(define LOS2 ('" "A" "BB"))

•

```
;; (listof X) (listof Y) → (listof XY)
;; Purpose: Create a new list with members of the first given
;;           list followed by members of the second given list
(define (myappend L1 L2)

;; Sample expressions for myappend
(define LON1-LOS1-VAL LOS1) append LON1 and LOS1
(define LON1-LON2-VAL LON2) append LON1 and LON2
(define LOS2-LOS1-VAL (cons (first LOS2) (myappend (rest LOS2) LOS1)))
(define LON2-LOS2-VAL (cons (first LON2) (myappend (rest LON2) LOS2)))
```


```

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- ```
;; Sample instances of (listof X), where X = number
(define LON1 '())
(define LON2 '(10 20 30 40))

;; Sample instances of (listof Y), where Y = string
(define LOS1 '())
(define LOS2 '("") "A" "BB"))

;; (listof X) (listof Y) → (listof XY)
;; Purpose: Create a new list with members of the first given
;;           list followed by members of the second given list
(define (myappend L1 L2)
```
- ```
;; Sample expressions for myappend
(define LON1-LOS1-VAL LOS1) append LON1 and LOS1
(define LON1-LON2-VAL LON2) append LON1 and LON2
(define LOS2-LOS1-VAL (cons (first LOS2) (myappend (rest LOS2) LOS1)))
(define LON2-LOS2-VAL (cons (first LON2) (myappend (rest LON2) LOS2)))

;; Tests using sample computations for myappend
(check-expect (myappend LON1 LOS1) LON1-LOS1-VAL)
(check-expect (myappend LON1 LON2) LON1-LON2-VAL)
(check-expect (myappend LOS2 LOS1) LOS2-LOS1-VAL)
(check-expect (myappend LON2 LOS2) LON2-LOS2-VAL)
;; Tests using sample values for myappend
(check-expect (myappend '#true #false) '(a b c))
 '#true #false a b c)
(check-expect (myappend '(M) '(T M)) '(M T M))
```

# Processing Multiple Inputs of Arbitrary Size

## One Input Has a Dominant Role

- ```
;; Sample instances of (listof X), where X = number
(define LON1 '())
(define LON2 '(10 20 30 40))
```
- ```
;; Sample instances of (listof Y), where Y = string
(define LOS1 '())
(define LOS2 ('" "A" "BB"))
```
- ```
;; (listof X) (listof Y) → (listof XY)
;; Purpose: Create a new list with members of the first given
;;           list followed by members of the second given list
(define (myappend L1 L2)
  (if (empty? L1)
      L2
      (cons (first L1) (myappend (rest L1) L2))))
```
- ```
;; Sample expressions for myappend
(define LON1-LOS1-VAL LOS1) append LON1 and LOS1
(define LON1-LON2-VAL LON2) append LON1 and LON2
(define LOS2-LOS1-VAL (cons (first LOS2) (myappend (rest LOS2) LOS1)))
(define LON2-LOS2-VAL (cons (first LON2) (myappend (rest LON2) LOS2)))
```
- ```
;; Tests using sample computations for myappend
(check-expect (myappend LON1 LOS1) LON1-LOS1-VAL)
(check-expect (myappend LON1 LON2) LON1-LON2-VAL)
(check-expect (myappend LOS2 LOS1) LOS2-LOS1-VAL)
(check-expect (myappend LON2 LOS2) LON2-LOS2-VAL)
;; Tests using sample values for myappend
(check-expect (myappend '#true #false) '(a b c))
  '#(true #false a b c))
(check-expect (myappend '(M) '(T M)) '(M T M))
```

Processing Multiple Inputs of Arbitrary Size

Homework

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- Problems: 213–214, 216

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.
- Cannot process one of the lists and simply return the other list when the first list is processed
- In other words, neither list is dominant
- Both lists must be processed at the same time

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.
- Cannot process one of the lists and simply return the other list when the first list is processed
- In other words, neither list is dominant
- Both lists must be processed at the same time
- As the lists are traversed the first element of both lists are multiplied and consed to the result

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.
- Cannot process one of the lists and simply return the other list when the first list is processed
- In other words, neither list is dominant
- Both lists must be processed at the same time
- As the lists are traversed the first element of both lists are multiplied and consed to the result
- The function is designed using the template for either input
- The recursive call must be made using the selector for the substructure of both inputs

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.
- Cannot process one of the lists and simply return the other list when the first list is processed
- In other words, neither list is dominant
- Both lists must be processed at the same time
- As the lists are traversed the first element of both lists are multiplied and consed to the result
- The function is designed using the template for either input
- The recursive call must be made using the selector for the substructure of both inputs
- Given that to multiply corresponding numbers the two lists must be of the same size, the lists are either both empty or they are both not empty
- What is the answer if the lists are empty?

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.
- Cannot process one of the lists and simply return the other list when the first list is processed
- In other words, neither list is dominant
- Both lists must be processed at the same time
- As the lists are traversed the first element of both lists are multiplied and consed to the result
- The function is designed using the template for either input
- The recursive call must be made using the selector for the substructure of both inputs
- Given that to multiply corresponding numbers the two lists must be of the same size, the lists are either both empty or they are both not empty
- What is the answer if the lists are empty?
- The result must be the empty list

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- Consider the problem of multiplying corresponding numbers in two lists of numbers and returning a list of the products
- The i^{th} elements of each list must be multiplied to compute a list of products
- Observe that both lists must be of the same size.
- Cannot process one of the lists and simply return the other list when the first list is processed
- In other words, neither list is dominant
- Both lists must be processed at the same time
- As the lists are traversed the first element of both lists are multiplied and consed to the result
- The function is designed using the template for either input
- The recursive call must be made using the selector for the substructure of both inputs
- Given that to multiply corresponding numbers the two lists must be of the same size, the lists are either both empty or they are both not empty
- What is the answer if the lists are empty?
- The result must be the empty list
- If the two lists are not empty then the product of first element of each list is consed to the result of multiplying the rest of both lists
- The function may be designed by specializing the template for a function on a `(listof number)`

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- ```
;; Sample instances of lon
(define ELON '())
(define HOURS-WORKED '(1 23 39 27))
(define HOURLY-RATES '(13 22 18 34)) (define QUANTITIES '(10 3 86 27 8))
(define PRICES '(80 50 5 10 20))
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Processing Multiple Inputs of Arbitrary Size

## Inputs Must Be Processed Simultaneously

- `;; Sample instances of lon`  
`(define ELON '())` `(define HOURS-WORKED '(1 23 39 27))`  
`(define HOURLY-RATES '(13 22 18 34))` `(define QUANTITIES '(10 3 86 27 8))`  
`(define PRICES '(80 50 5 10 20))`
- `;; Sample expressions for mlist`  
`(define NOTHING '())`  
`(define GROSSPAY`  
`(cons (* (first HOURS-WORKED) (first HOURLY-RATES))`  
`(mlist (rest HOURS-WORKED) (rest HOURLY-RATES))))`
- `(define TOTALS`  
`(cons (* (first QUANTITIES) (first PRICES))`  
`(mlist (rest QUANTITIES) (rest PRICES))))`

# Processing Multiple Inputs of Arbitrary Size

## Inputs Must Be Processed Simultaneously

- ```
;; Sample instances of lon
(define ELON '())
(define HOURS-WORKED '(1 23 39 27))
(define HOURLY-RATES '(13 22 18 34)) (define QUANTITIES '(10 3 86 27 8))
(define PRICES '(80 50 5 10 20))
```
- ```
;; (listof num) (listof num) → (listof num)
;; Purpose: Return a list with the products of corresponding
;; elements in the given lists
;; Assumption: The given lists are of the same length
(define (mlist L1 L2)
```
- ```
;; Sample expressions for mlist
(define NOTHING '())
(define GROSSPAY
  (cons (* (first HOURS-WORKED) (first HOURLY-RATES))
        (mlist (rest HOURS-WORKED) (rest HOURLY-RATES))))
(define TOTALS
  (cons (* (first QUANTITIES) (first PRICES))
        (mlist (rest QUANTITIES) (rest PRICES))))
```

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- ```
;; Sample instances of lon
(define ELON '())
(define HOURS-WORKED '(1 23 39 27))
(define HOURLY-RATES '(13 22 18 34)) (define QUANTITIES '(10 3 86 27 8))
(define PRICES '(80 50 5 10 20))
```
- ```
;; (listof num) (listof num) → (listof num)
;; Purpose: Return a list with the products of corresponding
;;           elements in the given lists
;; Assumption: The given lists are of the same length
(define (mlist L1 L2)
```
- ```
;; Sample expressions for mlist
(define NOTHING '())
(define GROSSPAY
 (cons (* (first HOURS-WORKED) (first HOURLY-RATES))
 (mlist (rest HOURS-WORKED) (rest HOURLY-RATES))))
(define TOTALS
 (cons (* (first QUANTITIES) (first PRICES))
 (mlist (rest QUANTITIES) (rest PRICES))))
```
- ```
; Tests using sample computations for mlist
(check-expect (mlist ELON      '())      NOTHING)
(check-expect (mlist HOURS-WORKED HOURLY-RATES) GROSSPAY)
(check-expect (mlist QUANTITIES  PRICES)      TOTALS)
;; Tests using sample values for mlist
(check-expect (mlist '(1 2 3) '(1 2 3)) '(1 4 9))
```

Processing Multiple Inputs of Arbitrary Size

Inputs Must Be Processed Simultaneously

- ```
;; Sample instances of lon
(define ELON '())
(define HOURS-WORKED '(1 23 39 27))
(define HOURLY-RATES '(13 22 18 34)) (define QUANTITIES '(10 3 86 27 8))
(define PRICES '(80 50 5 10 20))
```
- ```
;; (listof num) (listof num) → (listof num)
;; Purpose: Return a list with the products of corresponding
;;           elements in the given lists
;; Assumption: The given lists are of the same length
(define (mlist L1 L2)
```
- ```
(if (empty? L1)
 '()
 (cons (* (first L1) (first L2))
 (mlist (rest L1) (rest L2)))))
```
- ```
; Sample expressions for mlist
(define NOTHING '())
(define GROSSPAY
  (cons (* (first HOURS-WORKED) (first HOURLY-RATES))
        (mlist (rest HOURS-WORKED) (rest HOURLY-RATES))))
```
- ```
(define TOTALS
 (cons (* (first QUANTITIES) (first PRICES))
 (mlist (rest QUANTITIES) (rest PRICES))))
```
- ```
; Tests using sample computations for mlist
(check-expect (mlist ELON      '())      NOTHING)
(check-expect (mlist HOURS-WORKED HOURLY-RATES) GROSSPAY)
(check-expect (mlist QUANTITIES  PRICES)      TOTALS)
;; Tests using sample values for mlist
(check-expect (mlist '(1 2 3) '(1 2 3)) '(1 4 9))
```

Processing Multiple Inputs of Arbitrary Size

Homework

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- Problems: 217–220

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- It is not uncommon to have more than one complex input and not be able to identify a relationship between the inputs

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- It is not uncommon to have more than one complex input and not be able to identify a relationship between the inputs
- In such cases, it is necessary to analyze each possible combination of the inputs' subtypes
- A conditional expression is needed in which the number of conditions is equal to the number of possible input subtype combinations
- Each stanza in the conditional expression tests for a specific combination of subtypes.

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- It is not uncommon to have more than one complex input and not be able to identify a relationship between the inputs
- In such cases, it is necessary to analyze each possible combination of the inputs' subtypes
- A conditional expression is needed in which the number of conditions is equal to the number of possible input subtype combinations
- Each stanza in the conditional expression tests for a specific combination of subtypes.
- Consider a function that processes a (listof X) and a natural number
- Each has two subtypes
- There are four possible subtype combinations

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- It is not uncommon to have more than one complex input and not be able to identify a relationship between the inputs
- In such cases, it is necessary to analyze each possible combination of the inputs' subtypes
- A conditional expression is needed in which the number of conditions is equal to the number of possible input subtype combinations
- Each stanza in the conditional expression tests for a specific combination of subtypes.
- Consider a function that processes a (listof X) and a natural number
- Each has two subtypes
- There are four possible subtype combinations
- The needed conditional expression:

```
(cond [(and (empty? a-lox) (zero? a-natnum)) ...]  
      [(and (empty? a-lox) (not (zero? a-natnum))) ...]  
      [(and (cons? a-lox) (zero? a-natnum)) ...]  
      [else ...(f a-lox (sub1 n))...  
       ...(f (rest a-lox) n)...  
       ...(f (rest a-lox) (sub1 n))...])
```

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- Consider the problem of merging two lists of numbers sorted in nondecreasing order

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- Consider the problem of merging two lists of numbers sorted in nondecreasing order
- Ask yourself if one list dominates the other

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- Consider the problem of merging two lists of numbers sorted in nondecreasing order
- Ask yourself if one list dominates the other
- Clearly, this is not the case because the next element to add to the result list may come from either of the given lists.
- Ask you yourself if the processing of the lists must be synchronized

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- Consider the problem of merging two lists of numbers sorted in nondecreasing order
- Ask yourself if one list dominates the other
- Clearly, this is not the case because the next element to add to the result list may come from either of the given lists.
- Ask you yourself if the processing of the lists must be synchronized
- Once again, the answer is no because the first element of the result may come from either given list.

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- Consider the problem of merging two lists of numbers sorted in nondecreasing order
- Ask yourself if one list dominates the other
- Clearly, this is not the case because the next element to add to the result list may come from either of the given lists.
- Ask you yourself if the processing of the lists must be synchronized
- Once again, the answer is no because the first element of the result may come from either given list.
- We are left to conclude that there is no clear relationship between the inputs
- The needed conditional expression is outlined as follows:

```
(cond [(and (empty? s1) (empty? s2)) ...]  
      [(and (empty? s1) (cons? s2)) ...]  
      [(and (cons? s1) (empty? s2)) ...]  
      [else ...(merge s1 (rest s2))...  
       ...(merge (rest s1) s2)...  
       ...(merge (rest s1) (rest s2)) ...])])
```

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- (cond [(and (empty? s1) (empty? s2)) ...]
[(and (empty? s1) (cons? s2)) ...]
[(and (cons? s1) (empty? s2)) ...]
[else ...(merge s1 (rest s2))...
...(merge (rest s1) s2)...
...(merge (rest s1) (rest s2)) ...]]])

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ```
(cond [(and (empty? s1) (empty? s2)) ...]
 [(and (empty? s1) (cons? s2)) ...]
 [(and (cons? s1) (empty? s2)) ...]
 [else ...(merge s1 (rest s2))...
 ... (merge (rest s1) s2)...
 ... (merge (rest s1) (rest s2)) ...]]))
```
- Reason about each case in the conditional independently

# Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

## Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- ```
(cond [(and (empty? s1) (empty? s2)) ...]
        [(and (empty? s1) (cons? s2)) ...]
        [(and (cons? s1) (empty? s2)) ...]
        [else ...(merge s1          (rest s2))...
         ... (merge (rest s1) s2)...
         ... (merge (rest s1) (rest s2)) ...]])
```
- Reason about each case in the conditional independently
- If both given lists are empty then the answer is '()

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ```
(cond [(and (empty? s1) (empty? s2)) ...]
 [(and (empty? s1) (cons? s2)) ...]
 [(and (cons? s1) (empty? s2)) ...]
 [else ...(merge s1 (rest s2))...
 ...(merge (rest s1) s2)...
 ...(merge (rest s1) (rest s2)) ...]])
```
- Reason about each case in the conditional independently
- If both given lists are empty then the answer is '()
- If only one of the lists is empty then the answer is the other list

# Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

## Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- ```
(cond [(and (empty? s1) (empty? s2)) ...]
        [(and (empty? s1) (cons? s2)) ...]
        [(and (cons? s1) (empty? s2)) ...]
        [else ...(merge s1          (rest s2))...
         ...(merge (rest s1) s2)...
         ...(merge (rest s1) (rest s2)) ...]])
```
- Reason about each case in the conditional independently
- If both given lists are empty then the answer is '()
- If only one of the lists is empty then the answer is the other list
- If both lists are not empty then a decision must be made
- Either create a new list using s1's or s2's first element
- The element chosen must be the smaller of the two
- The recursive call, therefore, is made with the rest of the list whose first element is added to the result and the other list

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

- ```
;; Sample instances of (listof number)
(define ELON '()) (define SL1 '(1 22 30)) (define SL2 '(13 22 108 346))
(define SL3 '(-89 -50 0 6 90)) (define SL4 '(-240))
```

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

# Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- ```
;; Sample instances of (listof number)
(define ELON '())  (define SL1 '(1 22 30)) (define SL2 '(13 22 108 346))
(define SL3 '(-89 -50 0 6 90))  (define SL4 '(-240))
```

- ```
;; Sample expressions for merge
(define ELON-ELON-VAL '()) both lists empty
(define ELON-SL1-VAL SL1) one list empty
(define ELON-SL2-VAL SL2)
(define SL3-ELON-VAL SL3) (define SL4-ELON-VAL SL4)
(define SL1-SL2-VAL (cons (first SL1) (merge (rest SL1) SL2))) both lists not empty
(define SL3-SL2-VAL (cons (first SL3) (merge (rest SL3) SL2)))
(define SL3-SL4-VAL (cons (first SL4) (merge SL3 (rest SL4))))
(define SL1-SL4-VAL (cons (first SL4) (merge SL1 (rest SL4))))
```

# Processing Multiple Inputs of Arbitrary Size

## No Clear Relationship Between the Inputs

Lists

List  
Processing

Natural  
Numbers

Interval  
Processing

Aliens Attack  
Version 4

Binary Trees

Mutually  
Recursive  
Data

Processing  
Multiple  
Inputs of  
Arbitrary  
Size

- ```
;; Sample instances of (listof number)
(define ELON '())  (define SL1 '(1 22 30)) (define SL2 '(13 22 108 346))
(define SL3 '(-89 -50 0 6 90))  (define SL4 '(-240))
```
 - ```
;; (listof num) (listof num) → (listof num)
;; Purpose: Return a list sorted in nondecreasing order that
;; only contains all the elements of the given lists
;; Assumption: Given lists are sorted in nondecreasing order
(define (merge sl1 sl2))
```
- 
- ```
;; Sample expressions for merge
(define ELON-ELON-VAL '()) both lists empty
(define ELON-SL1-VAL SL1) one list empty
(define ELON-SL2-VAL SL2)
(define SL3-ELON-VAL SL3) (define SL4-ELON-VAL SL4)
(define SL1-SL2-VAL (cons (first SL1) (merge (rest SL1) SL2))) both lists not empty
(define SL3-SL2-VAL (cons (first SL3) (merge (rest SL3) SL2)))
(define SL3-SL4-VAL (cons (first SL4) (merge SL3 (rest SL4))))
(define SL1-SL4-VAL (cons (first SL4) (merge SL1 (rest SL4))))
```

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- `;; Sample instances of (listof number)`
`(define ELON '()) (define SL1 '(1 22 30)) (define SL2 '(13 22 108 346))`
`(define SL3 '(-89 -50 0 6 90)) (define SL4 '(-240))`
- `;; (listof num) (listof num) → (listof num)`
`;; Purpose: Return a list sorted in nondecreasing order that`
`;; only contains all the elements of the given lists`
`;; Assumption: Given lists are sorted in nondecreasing order`
`(define (merge sl1 sl2)`
- `(cond [(and (empty? sl1) (empty? sl2)) '()]`
`[(and (empty? sl1) (cons? sl2)) sl2]`
`[(and (cons? sl1) (empty? sl2)) sl1]`
`[else (if (<= (first sl1) (first sl2))`
`(cons (first sl1) (merge (rest sl1) sl2))`
`(cons (first sl2) (merge sl1 (rest sl2))))]))`
- `;; Sample expressions for merge`
`(define ELON-ELON-VAL '()) both lists empty`
`(define ELON-SL1-VAL SL1) one list empty`
`(define ELON-SL2-VAL SL2)`
`(define SL3-ELON-VAL SL3) (define SL4-ELON-VAL SL4)`
`(define SL1-SL2-VAL (cons (first SL1) (merge (rest SL1) SL2))) both lists not empty`
`(define SL3-SL2-VAL (cons (first SL3) (merge (rest SL3) SL2)))`
`(define SL3-SL4-VAL (cons (first SL4) (merge SL3 (rest SL4))))`
`(define SL1-SL4-VAL (cons (first SL4) (merge SL1 (rest SL4))))`

Processing Multiple Inputs of Arbitrary Size

No Clear Relationship Between the Inputs

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- ;; Tests using sample computations for merge
(check-expect (merge ELON ELON) ELON-ELON-VAL)
(check-expect (merge ELON SL1) ELON-SL1-VAL)
(check-expect (merge ELON SL2) ELON-SL2-VAL)
(check-expect (merge SL3 ELON) SL3-ELON-VAL)
(check-expect (merge SL4 ELON) SL4-ELON-VAL)
(check-expect (merge SL1 SL2) SL1-SL2-VAL)
(check-expect (merge SL3 SL2) SL3-SL2-VAL)
(check-expect (merge SL3 SL4) SL3-SL4-VAL)
(check-expect (merge SL1 SL4) SL1-SL4-VAL)

;; Tests using sample values for merge
(check-expect (merge '() '()) '())
(check-expect (merge '() '(8 77)) '(8 77))
(check-expect (merge '(43 67 91) '()) '(43 67 91))
(check-expect (merge '(15 31 67) '(22 44 87 100))
 '(15 22 31 44 67 87 100))
(check-expect (merge '(66 99) '(1 56 83)) '(1 56 66 83 99))

Processing Multiple Inputs of Arbitrary Size

Homework

Lists

List
Processing

Natural
Numbers

Interval
Processing

Aliens Attack
Version 4

Binary Trees

Mutually
Recursive
Data

Processing
Multiple
Inputs of
Arbitrary
Size

- Problems: 222–223