

FSequence ライブラリー

佐原伸日本フィット株式会社

情報技術研究所

TEL : 03-3623-4683

shin.sahara@jfits.co.jp

平成 16 年 3 月 5 日

概 要

列型に関わる関数を提供するモジュールである。

0.1 FSequence

列に関わる関数を提供する。列型で定義された機能以外の機能を定義する。

0.1.1 参照

多くの関数は、関数型プログラミング言語 Concurrent Clean や Standard ML のライブラリーから移植した。

class *FSequence*

Sum は列 *s* の要素の合計を返す。

functions

public static

```
1.0 Sum[@T] : @T* → @T
.1 Sum(s)  $\triangleq$ 
.2 Foldl[@T, @T] (Plus[@T]) (0) (s)
.3 pre is_(s, ℤ*) ∨ is_(s, ℕ*) ∨ is_(s, ℕ1*) ∨
.4 is_(s, ℝ*) ∨
.5 is_(s, ℚ*) ;
```

Prod は列 *s* の全要素の積を返す。

public static

```
2.0 Prod[@T] : @T* → @T
.1 Prod(s)  $\triangleq$ 
.2 Foldl[@T, @T] (Product[@T]) (1) (s)
.3 pre is_(s, ℤ*) ∨ is_(s, ℕ*) ∨ is_(s, ℕ1*) ∨
.4 is_(s, ℝ*) ∨
.5 is_(s, ℚ*) ;
```

Plus は加算を行う。

public static

```
3.0 Plus[@T] : @T → @T → @T
.1 Plus(a)(b)  $\triangleq$ 
.2 a + b;
```

Product は積算を行う。

public static

```
4.0 Product[@T] : @T → @T → @T
.1 Product(a)(b)  $\triangleq$ 
.2 a × b;
```

Append は列の追加を行う。

public static

```
5.0 Append[@T] : @T* → @T → @T*  
.1 Append (s) (e)  $\triangle$   
.2  $s \curvearrowright [e];$ 
```

Average は列 *s* の要素の平均を求める。

public static

```
6.0 Average[@T] : @T* → [ℝ]  
.1 Average (s)  $\triangle$   
.2 if  $s = []$   
.3 then nil  
.4 else AverageAux[@T] (0) (0) (s) post if  $s = []$   
.5 then RESULT = nil  
.6 else RESULT = Sum[@T] (s) / len s;  
  
7.0 AverageAux[@T] : @T → @T → @T* → ℝ  
.1 AverageAux (total) (numOfElem) (s)  $\triangle$   
.2 cases s :  
.3  $[x] \curvearrowright xs \rightarrow \textit{AverageAux}[@T] (total + x) (numOfElem + 1) (xs),  
.4  $[] \rightarrow \textit{total} / \textit{numOfElem}$   
.5 end;$ 
```

IsAscendingInTotalOrder は、関数 *f* で与えられた全順序で、列 *s* の要素が昇順であるか否かを返す。

public static

```
8.0 IsAscendingInTotalOrder[@T] : (@T × @T → ℬ) → @T* → ℬ  
.1 IsAscendingInTotalOrder (f) (s)  $\triangle$   
.2  $\forall i, j \in \text{inds } s \cdot i < j \Rightarrow f(s(i), s(j)) \vee s(i) = s(j);$ 
```

IsDescendingInTotalOrder は、関数 *f* で与えられた全順序で、列 *s* の要素が降順であるか否かを返す。

public static

```
9.0 IsDescendingInTotalOrder[@T] : (@T × @T → ℬ) → @T* → ℬ  
.1 IsDescendingInTotalOrder (f) (s)  $\triangle$   
.2  $\forall i, j \in \text{inds } s \cdot i < j \Rightarrow f(s(j), s(i)) \vee s(i) = s(j);$ 
```

IsAscending は、演算子 *i* で与えられた全順序で、列 *s* の要素が昇順であるか否かを返す。

public static

```

10.0  IsAscending[@T] : @T* → ℬ
      .1  IsAscending (s)  $\triangleq$ 
      .2  IsAscendingInTotalOrder[@T] (λ x : @T, y : @T · x < y) (s);

```

IsDescending は、演算子_iで与えられた全順序で、列 s の要素が降順であるか否かを返す。

public static

```

11.0  IsDescending[@T] : @T* → ℬ
      .1  IsDescending (s)  $\triangleq$ 
      .2  IsDescendingInTotalOrder[@T] (λ x : @T, y : @T · x < y) (s);

```

Sort は、関数 f で与えられた順序で、列 s の要素を昇順にクイックソートする。

public static

```

12.0  Sort[@T] : (@T × @T → ℬ) → @T* → @T*
      .1  Sort (f)(s)  $\triangleq$ 
      .2  cases s :
      .3  [] → [],
      .4  [x]  $\curvearrowright$  xs →
      .5      Sort[@T] (f) ([xs (i) | i ∈ inds xs · f (xs (i), x)])  $\curvearrowright$ 
      .6      [x]  $\curvearrowright$ 
      .7      Sort[@T] (f) ([xs (i) | i ∈ inds xs · ¬ f (xs (i), x)])
      .8  end;

```

AscendingSort は、演算子_iで与えられた順序で、列 s の要素を昇順にソートする。

public static

```

13.0  AscendingSort[@T] : @T* → @T*
      .1  AscendingSort (s)  $\triangleq$ 
      .2  Sort[@T] (λ x : @T, y : @T · x < y) (s) post IsAscending[@T] (RESULT);

```

DescendingSort は、演算子_iで与えられた順序で、列 s の要素を昇順にソートする。演算子_iから見れば降順。

public static

```

14.0  DescendingSort[@T] : @T* → @T*
      .1  DescendingSort (s)  $\triangleq$ 
      .2  Sort[@T] (λ x : @T, y : @T · x > y) (s) post IsDescending[@T] (RESULT);

```

IsOrdered は、順序決定関数列 fs で与えられた順序であれば true、そうでなければ false を返す。

public static

```

15.0  $IsOrdered[@T] : (@T \times @T \rightarrow \mathbb{B})^* \rightarrow @T^* \rightarrow @T^* \rightarrow \mathbb{B}$ 
.1  $IsOrdered(f)(s1)(s2) \triangleq$ 
.2   cases mk- (s1, s2) :
.3     mk- ([], [])  $\rightarrow$  false,
.4     mk- ([], -)  $\rightarrow$  true,
.5     mk- (-, [])  $\rightarrow$  false,
.6     mk- ([x1]  $\curvearrowright$  xs1, [x2]  $\curvearrowright$  xs2)  $\rightarrow$ 
.7       if (hd f) (x1, x2)
.8       then true
.9       elseif (hd f) (x2, x1)
.10      then false
.11      else  $IsOrdered[@T](tl\ f)(xs1)(xs2)$ 
.12   end;

```

Merge は、関数 f で与えられた順序で、列 s1,s2 の要素をマージする。
public static

```

16.0  $Merge[@T] : (@T \times @T \rightarrow \mathbb{B}) \rightarrow @T^* \rightarrow @T^* \rightarrow @T^*$ 
.1  $Merge(f)(s1)(s2) \triangleq$ 
.2   cases mk- (s1, s2) :
.3     mk- ([], y)  $\rightarrow$  y,
.4     mk- (x, [])  $\rightarrow$  x,
.5     mk- ([x1]  $\curvearrowright$  xs1, [x2]  $\curvearrowright$  xs2)  $\rightarrow$ 
.6       if f (x1, x2)
.7       then [x1]  $\curvearrowright$   $FSequence\ Merge[@T](f)(xs1)(s2)$ 
.8       else [x2]  $\curvearrowright$   $FSequence\ Merge[@T](f)(s1)(xs2)$ 
.9   end;

```

InsertAt は、列 s の指定された位置 i に要素 e を追加する。
public static

```

17.0  $InsertAt[@T] : \mathbb{N}_1 \rightarrow @T \rightarrow @T^* \rightarrow @T^*$ 
.1  $InsertAt(i)(e)(s) \triangleq$ 
.2   cases mk- (i, s) :
.3     mk- (1, s)  $\rightarrow$  [e]  $\curvearrowright$  s,
.4     mk- (-, [])  $\rightarrow$  [e],
.5     mk- (i, [x]  $\curvearrowright$  xs)  $\rightarrow$  [x]  $\curvearrowright$   $InsertAt[@T](i-1)(e)(xs)$ 
.6   end;

```

RemoveAt は、列 s の指定された位置 i の要素を削除する。
public static

```

18.0  RemoveAt[@T] :  $\mathbb{N}_1 \rightarrow @T^* \rightarrow @T^*$ 
.1    RemoveAt(i)(s)  $\triangle$ 
.2      cases mk-(i, s) :
.3        mk-(1, [-]  $\curvearrowright$  xs)  $\rightarrow$  xs,
.4        mk-(i, [x]  $\curvearrowright$  xs)  $\rightarrow$  [x]  $\curvearrowright$  RemoveAt[@T](i - 1)(xs),
.5        mk-(-, [])  $\rightarrow$  []
.6      end;

```

RemoveDup は、列 s から重複する要素を削除する。
public static

```

19.0  RemoveDup[@T] : @T*  $\rightarrow$  @T*
.1    RemoveDup(s)  $\triangle$ 
.2      cases s :
.3        [x]  $\curvearrowright$  xs  $\rightarrow$  [x]
RemoveDup[@T](Filter[@T]( $\lambda e : @T \cdot e \neq x$ )(xs)),
.4        []  $\rightarrow$  []
.5      endpost  $\neg$  IsDup[@T](RESULT);

```

RemoveMember は、列 s から要素 e を削除する。
public static

```

20.0  RemoveMember[@T] : @T  $\rightarrow$  @T*  $\rightarrow$  @T*
.1    RemoveMember(e)(s)  $\triangle$ 
.2      cases s :
.3        [x]  $\curvearrowright$  xs  $\rightarrow$  if e = x
.4                          then xs
.5                          else [x]  $\curvearrowright$  RemoveMember[@T](e)(xs),
.6        []  $\rightarrow$  []
.7      end;

```

RemoveMembers は、列 s から要素列 es の全要素を削除する。
public static

```

21.0  RemoveMembers[@T] : @T*  $\rightarrow$  @T*  $\rightarrow$  @T*
.1    RemoveMembers(es)(s)  $\triangle$ 
.2      cases es :
.3        []  $\rightarrow$  s,
.4        [x]  $\curvearrowright$  xs  $\rightarrow$  RemoveMembers[@T](xs)(RemoveMember[@T](x)(s))
.5      end;

```

UpdateAt は、列 s の指定された位置 i の要素 e を指定された新要素で置き換える。
public static

```

22.0  UpdateAt[@T] :  $\mathbb{N}_1 \rightarrow @T \rightarrow @T^* \rightarrow @T^*$ 
.1    UpdateAt (i)(e)(s)  $\triangleq$ 
.2      cases mk- (i, s) :
.3        mk- (-, [])  $\rightarrow$  [],
.4        mk- (1, [-]  $\curvearrowright$  xs)  $\rightarrow$  [e]  $\curvearrowright$  xs,
.5        mk- (i, [x]  $\curvearrowright$  xs)  $\rightarrow$  [x]  $\curvearrowright$  UpdateAt[@T] (i - 1) (e) (xs)
.6      end;

```

Take は、列 s の先頭 i 個からなる列を返す。

public static

```

23.0  Take[@T] :  $\mathbb{Z} \rightarrow @T^* \rightarrow @T^*$ 
.1    Take (i)(s)  $\triangleq$ 
.2      s (1, ..., i);

```

TakeWhile は、列 s の先頭から、関数 f を満たし続ける間の部分列を返す。

public static

```

24.0  TakeWhile[@T] : ( $@T \rightarrow \mathbb{B}$ )  $\rightarrow @T^* \rightarrow @T^*$ 
.1    TakeWhile (f)(s)  $\triangleq$ 
.2      cases s :
.3        [x]  $\curvearrowright$  xs  $\rightarrow$ 
.4          if f (x)
.5            then [x]  $\curvearrowright$  TakeWhile[@T] (f) (xs)
.6            else [],
.7        []  $\rightarrow$  []
.8      end;

```

Drop は、列 s の先頭 i 個を除く列を返す。

public static

```

25.0  Drop[@T] :  $\mathbb{Z} \rightarrow @T^* \rightarrow @T^*$ 
.1    Drop (i)(s)  $\triangleq$ 
.2      s (i + 1, ..., len s);

```

DropWhile は、列 s の先頭から、関数 f を満たさない間の部分列を返す。

public static

26.0 $DropWhile[@T] : (@T \rightarrow \mathbb{B}) \rightarrow @T^* \rightarrow @T^*$

```
.1  $DropWhile(f)(s) \triangleq$ 
.2   cases  $s$  :
.3      $[x] \curvearrowright xs \rightarrow$ 
.4       if  $f(x)$ 
.5       then  $DropWhile[@T](f)(xs)$ 
.6       else  $s$ ,
.7      $[] \rightarrow []$ 
.8   end;
```

Span は、指定された列 s を、先頭から関数 f を満たし続ける間の列と、関数を満たさなくなつて以降の列の組に分ける。

public static

27.0 $Span[@T] : (@T \rightarrow \mathbb{B}) \rightarrow @T^* \rightarrow @T^* \times @T^*$

```
.1  $Span(f)(s) \triangleq$ 
.2   cases  $s$  :
.3      $[x] \curvearrowright xs \rightarrow$ 
.4       if  $f(x)$ 
.5       then let  $mk-(satisfied, notSatisfied) = Span[@T](f)(xs)$  in
.6          $mk-([x] \curvearrowright satisfied, notSatisfied)$ 
.7       else  $mk-([], s)$ ,
.8      $[] \rightarrow mk-([], [])$ 
.9   end;
```

SubSeq は、列 s の開始位置 i から要素数分取り出した部分列を返す

public static

28.0 $SubSeq[@T] : \mathbb{N} \rightarrow \mathbb{N} \rightarrow @T^+ \rightarrow @T^*$

```
.1  $SubSeq(i)(numOfElems)(s) \triangleq$ 
.2    $Take[char](numOfElems)(Drop[char](i-1)(s));$ 
```

Last は、列 s の最後の要素を返す。

public static

29.0 $Last[@T] : @T^* \rightarrow @T$

```
.1  $Last(s) \triangleq$ 
.2    $s(\text{len } s);$ 
```

Fmap は、関数を列に適用した結果の列を返す。

public static

30.0 $Fmap[@T1, @T2] : (@T1 \rightarrow @T2) \rightarrow @T1^* \rightarrow @T2^*$

```
.1  $Fmap(f)(s) \triangleq$ 
.2    $[f(s(i)) \mid i \in \text{inds } s];$ 
```


Filter は、指定された関数 f によって列 s を濾過する。つまり、列のうち関数を満たすものの列を返す。

public static

```
31.0  Filter[@T] : (@T → ℬ) → @T* → @T*
      .1  Filter(f)(s) ≜
      .2    [s(i) | i ∈ inds s · f(s(i))];
```

Foldl は、列 s に対するたたみ込み演算（関数 f を列 s の左側から適用）

public static

```
32.0  Foldl[@T1, @T2] : (@T1 → @T2 → @T1) → @T1 → @T2*
      → @T1
      .1  Foldl(f)(args)(s) ≜
      .2    cases s :
      .3      [] → args,
      .4      [x] ↷ xs → Foldl[@T1, @T2](f)(f(args)(x))(xs)
      .5    end;
```

Foldr は、列 s に対するたたみ込み演算（関数 f を列 s の右側から適用）

public static

```
33.0  Foldr[@T1, @T2] : (@T1 → @T2 → @T2) → @T2 → @T1*
      → @T2
      .1  Foldr(f)(args)(s) ≜
      .2    cases s :
      .3      [] → args,
      .4      [x] ↷ xs → f(x)(Foldr[@T1, @T2](f)(args)(xs))
      .5    end;
```

IsMember は、要素があるか否かを返す。

public static

```
34.0  IsMember[@T] : @T → @T* → ℬ
      .1  IsMember(e)(s) ≜
      .2    cases s :
      .3      [x] ↷ xs → e = x ∨ IsMember[@T](e)(xs),
      .4      [] → false
      .5    end;
```

IsAnyMember は、要素列 es 中の要素が、列 s にあるか否かを返す。

public static

```

35.0  IsAnyMember[@ T] : @ T* → @ T* →  $\mathbb{B}$ 
      .1  IsAnyMember (es)(s)  $\triangleq$ 
      .2    cases es :
      .3      [x]  $\curvearrowright$  xs → IsMember[@ T] (x) (s)
IsAnyMember[@ T] (xs) (s),
      .4      [] → false
      .5    end;

```

IsDup は、列 *s* 中に同じ要素があるか否かを返す。
public static

```

36.0  IsDup[@ T] : @ T* →  $\mathbb{B}$ 
      .1  IsDup (s)  $\triangleq$ 
      .2     $\neg$  card elems s = len spost if s = []
      .3      then RESULT = false
      .4      else RESULT =  $\neg \forall i, j \in \text{inds } s \cdot i \neq j \Rightarrow s(i) \neq s(j)$ ;

```

Index は、指定された要素 *e* が列 *s* の何番目にあるかを返す。最初の要素の位置を返す。
public static

```

37.0  Index[@ T] : @ T → @ T* →  $\mathbb{Z}$ 
      .1  Index (e)(s)  $\triangleq$ 
      .2    let i = 0 in
      .3      IndexAux[@ T] (e) (s) (i);

```

public static

```

38.0  IndexAux[@ T] : @ T → @ T* →  $\mathbb{Z}$  →  $\mathbb{Z}$ 
      .1  IndexAux (e)(s)(i)  $\triangleq$ 
      .2    cases s :
      .3      [] → 0,
      .4      [x]  $\curvearrowright$  xs →
      .5        if x = e
      .6        then i + 1
      .7        else IndexAux[@ T] (e) (xs) (i + 1)
      .8    end;

```

IndexAll は、指定された要素 *e* が列 *s* の何番目にあるかを持つ自然数集合を返す。
public static

```

39.0  IndexAll[@ T] : @ T → @ T* →  $\mathbb{N}_1$ -set
      .1  IndexAll (e)(s)  $\triangleq$ 
      .2    {i | i ∈ inds s · s (i) = e};

```

Flatten は、列 *s* の要素が列の場合、その要素を要素として持つ列を返す。

public static

40.0 $Flatten[@T] : @T^{**} \rightarrow @T^*$
.1 $Flatten(s) \triangleq$
.2 $\text{conc } s;$

Compact は、列 s の要素が nil のものを削除した列を返す
public static

41.0 $Compact[@T] : [@T]^* \rightarrow @T^*$
.1 $Compact(s) \triangleq$
.2 $\text{nil }]\text{post } \forall i \in \text{inds } RESULT \cdot RESULT(i) \neq \text{nil};$
$$[s(i) \mid i \in \text{inds } s \cdot s(i) \neq \text{nil}]$$

Freverse は、列 s の逆順の列を得る。reverse が予約語のため、Freverse という関数名にした。
public static

42.0 $Freverse[@T] : @T^* \rightarrow @T^*$
.1 $Freverse(s) \triangleq$
.2 $[s(\text{len } s + 1 - i) \mid i \in \text{inds } s];$

Permutations は、列 s から順列を得る
public static

43.0 $Permutations[@T] : @T^* \rightarrow @T^*\text{-set}$
.1 $Permutations(s) \triangleq$
.2 $\text{cases } s :$
.3 $[], [-] \rightarrow \{s\},$
.4 $\text{others} \rightarrow \bigcup \{ \{ [s(i)] \circ j \mid j \in \text{inds } s \} \mid i \in \text{inds } s \}$
.5 $\text{endpost } \forall x \in RESULT \cdot \text{elems } x = \text{elems } s;$

IsPermutations は、列 s が列 t の置換になっているか否かを返す。
public static

44.0 $IsPermutations[@T] : @T^* \rightarrow @T^* \rightarrow \mathbb{B}$
.1 $IsPermutations(s)(t) \triangleq$
.2 $\text{RemoveMembers}[@T](s)(t) = \text{RemoveMembers}[@T](t)(s);$

Unzip は、組の列を、列の組に変換する
public static

```

45.0  Unzip[@ T1, @ T2] : (@ T1 × @ T2)* → @ T1* × @ T2*
.1    Unzip (s)  $\triangle$ 
.2    cases s :
.3      [] → mk- ([], []),
.4      [mk- (x, y)]  $\curvearrowright$  xs →
.5          let mk- (s, t) = Unzip[@ T1, @ T2] (xs) in
.6          mk- ([x]  $\curvearrowright$  s, [y]  $\curvearrowright$  t)
.7    end;

```

Zip は、列の組を、組の列に変換する

public static

```

46.0  Zip[@ T1, @ T2] : @ T1* × @ T2* → (@ T1 × @ T2)*
.1    Zip (s1, s2)  $\triangle$ 
.2    Zip2[@ T1, @ T2] (s1) (s2);

```

Zip2 は、列の組を、組の列に変換する（より関数型プログラミングに適した形式）

public static

```

47.0  Zip2[@ T1, @ T2] : @ T1* → @ T2* → (@ T1 × @ T2)*
.1    Zip2 (s1)(s2)  $\triangle$ 
.2    cases mk- (s1, s2) :
.3      mk- ([x1]  $\curvearrowright$  xs1, [x2]  $\curvearrowright$  xs2) → [mk- (x1, x2)]
Zip2[@ T1, @ T2] (xs1) (xs2),
.4      mk- (-, -) → []
.5    end

```

end FSequence

Test Suite : vdm.tc

Class : FSequence

Name	#Calls	Coverage
FSequence'Sum	4	53%
FSequence'Zip	4	83%
FSequence'Drop	2	✓
FSequence'Fmap	3	✓
FSequence'Last	1	✓
FSequence'Plus	24	✓
FSequence'Prod	5	53%
FSequence'Sort	35	95%
FSequence'Span	14	96%

Name	#Calls	Coverage
FSequence‘Take	4	✓
FSequence‘Zip2	16	93%
FSequence‘Foldl	48	93%
FSequence‘Foldr	12	93%
FSequence‘Index	171	90%
FSequence‘IsDup	4	22%
FSequence‘Merge	16	94%
FSequence‘Unzip	5	95%
FSequence‘Append	3	✓
FSequence‘Filter	9	✓
FSequence‘SubSeq	1	84%
FSequence‘Average	3	41%
FSequence‘Compact	3	57%
FSequence‘Flatten	1	✓
FSequence‘Product	15	✓
FSequence‘Freverse	2	✓
FSequence‘IndexAll	4	✓
FSequence‘IndexAux	2717	95%
FSequence‘InsertAt	20	96%
FSequence‘IsMember	39	92%
FSequence‘RemoveAt	62	94%
FSequence‘UpdateAt	22	95%
FSequence‘DropWhile	6	93%
FSequence‘IsOrdered	15	97%
FSequence‘RemoveDup	11	70%
FSequence‘TakeWhile	6	94%
FSequence‘AverageAux	9	94%
FSequence‘IsAnyMember	6	88%
FSequence‘IsAscending	2	88%
FSequence‘IsDescending	1	88%
FSequence‘Permutations	24	71%
FSequence‘RemoveMember	86	94%
FSequence‘AscendingSort	1	61%

Name	#Calls	Coverage
FSequence'RemoveMembers	62	86%
FSequence'DescendingSort	1	61%
FSequence'IsPermutations	6	88%
FSequence'IsAscendingInTotalOrder	2	✓
FSequence'IsDescendingInTotalOrder	3	✓
Total Coverage		84%

0.2 FSequenceT

FSequence のテストを行う。

class *FSequenceT*

types

48.0 public *TestType* = \mathbb{Z} | *char** | *char*;

public

49.0 *Record* :: *v* : \mathbb{Z}

.1 *str* : *char**

.2 *c* : *char*

functions

public static

50.0 *run* : () → \mathbb{B}^*

.1 *run* () \triangleq

.2 let *testcases* =

.3 [

.4 *t1* (), *t2* (), *t3* (), *t4* (), *t5* (), *t6* (), *t7* (), *t8* (), *t9* (), *t10* (),

.5 *t11* (), *t12* (), *t13* (), *t14* (), *t15* (), *t16* (), *t17* (), *t18* (), *t19* (), *t20* (),

.6 *t21* (), *t22* (), *t23* (), *t24* ()] in

.7 *FSequence*'*Fmap*[*FTestDriver*'*TestCase**, \mathbb{B}] (*FTestDriver*'*run*) (*testcases*);

0.2.1 合計と積を検査する

51.0 *t1* : () → *FTestDriver*'*TestCase**

.1 *t1* () \triangleq

.2 [

.3 mk-*FTestDriver*'*TestCase*

.4 (

.5 "*FSequenceT01*

:"

\t254088A0830687A4D3092691C67FB3059308B",

.6 *FSequence*'*Sum*[\mathbb{Z}] ([1, 2, 3, 4, 5, 6, 7, 8, 9]) = 45 ∧

.7 *FSequence*'*Sum*[\mathbb{Z}] ([]) = 0 ∧

.8 *FSequence*'*Prod*[\mathbb{Z}] ([2, 3, 4]) = 24 ∧

.9 *FSequence*'*Prod*[\mathbb{Z}] ([]) = 1 ∧

.10 *FSequence*'*Sum*[\mathbb{R}] ([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]) =
4.5 ∧

.11 *FSequence*'*Sum*[\mathbb{R}] ([]) = 0 ∧

.12 *FSequence*'*Prod*[\mathbb{R}] ([2, 3, 4]) = 24 ∧

.13 *FSequence*'*Prod*[\mathbb{R}] ([]) = 1 ∧

.14 *FSequence*'*Prod*[\mathbb{R}] ([2.1, 3.2, 4.3]) = 2.1 × 3.2 × 4.3];

0.2.2 全順序昇順か？を検査する

```

52.0  t2 : () → FTestDriver `TestCase*
.1    t2 ()  $\triangle$ 
.2    [
.3      mk-FTestDriver `TestCase
.4      (
.5        "FSequenceT02"
\ t2516898065E8F66079806304B1F3092691C67FB3059308B",
.6        FSequence `IsAscending[ $\mathbb{Z}$ ] ([1, 2, 4, 4, 7, 8, 8, 8])  $\wedge$ 
.7         $\neg$  FSequence `IsAscending[ $\mathbb{R}$ ] ([1, 2, 3, 1.5]));

```

0.2.3 全順序降順か？を検査する

```

53.0  t3 : () → FTestDriver `TestCase*
.1    t3 ()  $\triangle$ 
.2    [
.3      mk-FTestDriver `TestCase
.4      (
.5        "FSequenceT03"
\ t2516898065E8F964D9806304B1F3092691C67FB3059308B",
.6        FSequence `IsDescending[ $\mathbb{Z}$ ] ([3, 2, 2, 1, 1])  $\wedge$ 
.7        FSequence `IsDescendingInTotalOrder[ $\mathbb{Z}$ ] ( $\lambda x : \mathbb{Z}, y : \mathbb{Z} \cdot x < y$ ) ([3, 2, 2, 1, 1])  $\wedge$ 
.8        FSequence `IsDescendingInTotalOrder[ $\mathbb{Z}$ ] ( $\lambda x : \mathbb{Z}, y : \mathbb{Z} \cdot x < y$ ) ([3, 2, 2, 1, 2]) =
false)];

```

0.2.4 順番になっているか？を検査する


```

54.0  t4 : () → FTestDriver' TestCase*
.1    t4 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT04
\29806756A306B306A306330663044308B304B1F3092691C67FB3059308B",
.6        let sq = new FSequence()
.7          fs =
.8            [(\x : ℤ, y : ℤ · x < y),
.9              \x : char*, y : char* · FString' LT (x) (y),
.10             \x : char, y : char · FCharacter' LT (x) (y)],
.11          f =
.12            \x : Record, y : Record ·
.13              FSequence' IsOrdered[TestType] .14
.16          FSequence' Sort[Record] (f)
.17          (
.18            [mk-Record (10, "sahara", 'c'), mk-Record (10, "sahara", 'a')] =
.19            [mk-Record (10, "sahara", 'a'), mk-Record (10, "sahara", 'c')] ∧
.20            sq.IsOrdered (fs) ([3, "123", 'a']) ([3, "123", 'A']) =
true ∧
.21            sq.IsOrdered (fs) ([3, "123", 'a']) ([3, "123", '0']) =
false ∧
.22            sq.IsOrdered (fs) ([3, "123", '0']) ([3, "123", 'A']) = false ∧
.23            sq.IsOrdered (fs) ([3, "123", '0']) ([3, "123", '0']) = true ∧
.24            sq.IsOrdered (fs) ([3, "123", '0']) ([3, "123", 'A']) = false];

```

0.2.5 マージを検査する

```

55.0  t5 : () → FTestDriver' TestCase*
.1    t5() △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT05" :
\ t230DE30FC30B83092691C67FB3059308B",
.6        let sq = new FSequence()
.7          f1 = λ x : ℤ, y : ℤ · x < y,
.8          f2 = λ x : char, y : char · FCharacter' LT (x) (y) in
.9          sq.Merge (f1) ([1, 4, 6]) ([2, 3, 4, 5]) = [1, 2, 3, 4, 4, 5, 6] ∧
.10         sq.Merge (f2) ("146") ("2345") = "1234456" ∧
.11         sq.Merge (f2) ("") ("2345") = "2345" ∧
.12         sq.Merge (f2) ("146") ("") = "146"]];

```

0.2.6 文字列操作を検査する

```

56.0  t6 : () → FTestDriver' TestCase*
.1    t6() △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT06" :
\ t265875B57521764CD4F5C3092691C67FB3059308B",
.6        let sq = new FSequence()
.7          sq.Take (2) ([2, 3, 4, 5]) = [2, 3] ∧
.8          sq.Drop (5) ("Shin2Sahara") = "Sahara" ∧
.9          sq.Last ([1, 2, 3]) = 3 ∧
.10         sq.Filter (λ x : ℤ · x mod 2 = 0) ([1, 2, 3, 4, 5, 6]) =
[2, 4, 6] ∧
.11         FSequence' SubSeq[char] (4) (3) ("1234567890") =
"456" ∧
.12         FSequence' Flatten[ℤ] ([[1, 2, 3], [3, 4], [4, 5, 6]]) =
[1, 2, 3, 3, 4, 4, 5, 6]]];

```

0.2.7 ソートを検査する

```

57.0  t7 : () → FTestDriver' TestCase*
      .1  t7 () △
      .2  [
      .3    mk-FTestDriver' TestCase
      .4    (
      .5      "FSequenceT07                                     :
      \t230BD30FC30C83092691C67FB3059308B",
      .6      FSequence' AscendingSort[ℤ] ([3, 1, 6, 4, 2, 6, 5]) =
      [1, 2, 3, 4, 5, 6, 6] ∧
      .7      FSequence' DescendingSort[ℤ] ([3, 1, 6, 4, 2, 6, 5]) =
      [6, 6, 5, 4, 3, 2, 1]);

```

0.2.8 空要素削除を検査する

```

58.0  t8 : () → FTestDriver' TestCase*
      .1  t8 () △
      .2  [
      .3    mk-FTestDriver' TestCase
      .4    (
      .5      "FSequenceT08                                     :
      \t27A7A89817D20524A96643092691C67FB3059308B",
      .6      FSequence' Compact[[ℤ]] ([3, 1, 6, 4, nil , 2, 6, 5, nil ]) =
      [3, 1, 6, 4, 2, 6, 5] ∧
      .7      FSequence' Compact[[ℤ]] ([nil , nil ]) = [] ∧
      .8      FSequence' Compact[[ℤ]] ([ ]) = [ ]);

```

0.2.9 列の反転を検査する

```

59.0  t9 : () → FTestDriver' TestCase*
      .1  t9 () △
      .2  [
      .3    mk-FTestDriver' TestCase
      .4    (
      .5      "FSequenceT09                                     :
      \t25217306E53CD8EE23092691C67FB3059308B",
      .6      FSequence' Freverse[[ℤ]] ([3, 1, 6, 4, nil , 2, 6, 5, nil ]) =
      [nil , 5, 6, 2, nil , 4, 6, 1, 3] ∧
      .7      FSequence' Freverse[[ℤ]] ([ ]) = [ ]);

```

0.2.10 順列を検査する

```

60.0  t10 : () → FTestDriver' TestCase*
.1    t10 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT10
\ t2980652173092691C67FB3059308B",
.6        FSequence' Permutations[[Z]] ([1, 2, 3]) =
.7        { [1, 2, 3],
.8          [1, 3, 2],
.9          [2, 1, 3],
.10         [2, 3, 1],
.11         [3, 1, 2],
.12         [3, 2, 1] } ∧
.13        FSequence' Permutations[[Z]] ([1, 2, 2]) =
.14        { [1, 2, 2],
.15          [2, 1, 2],
.16          [2, 2, 1] } ∧
.17        FSequence' Permutations[[B]] ([true, false]) =
.18        { [true, false],
.19          [false, true] } ∧
.20        FSequence' Permutations[[Z]] ([]) = { [] } ∧
.21        FSequence' IsPermutations[Z] ([1, 2, 3]) ([1, 3, 2]) ∧
.22        FSequence' IsPermutations[Z] ([1, 2, 3]) ([2, 1, 3]) ∧
.23        FSequence' IsPermutations[Z] ([1, 2, 3]) ([2, 3, 1]) ∧
.24        FSequence' IsPermutations[Z] ([1, 2, 3]) ([3, 1, 2]) ∧
.25        FSequence' IsPermutations[Z] ([1, 2, 3]) ([3, 2, 1]) ∧
.26        FSequence' IsPermutations[Z] ([1, 2, 3]) ([3, 2, 2]) =
false)];

```

0.2.11 列の要素か？を検査する

```

61.0  t11 : () → FTestDriver' TestCase*
.1    t11 () ≜
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT11" :
\ t25217306E89817D20304B1F3092691C67FB3059308B",
.6        FSequence' IsMember $\mathbb{Z}$  (2) ([1, 2, 3, 4, 5, 6]) ∧
.7        FSequence' IsMember $\mathbb{Z}$  (0) ([1, 2, 3, 4, 5, 6]) = false ∧
.8        FSequence' IsMember $\mathbb{Z}$  (6) ([1, 2, 3, 4, 5, 6]) ∧
.9        FSequence' IsAnyMember $\mathbb{Z}$  ([6]) ([1, 2, 3, 4, 5, 6]) ∧
.10       FSequence' IsAnyMember $\mathbb{Z}$  ([0, 7]) ([1, 2, 3, 4, 5, 6]) =
false ∧
.11       FSequence' IsAnyMember $\mathbb{Z}$  ([4, 6]) ([1, 2, 3, 4, 5, 6]) ∧
.12       FSequence' IsAnyMember $\mathbb{Z}$  ([]) ([1, 2, 3, 4, 5, 6]) =
false)];

```

0.2.12 Fmap を検査する

```

62.0  t12 : () → FTestDriver' TestCase*
.1    t12 () ≜
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT12 : \ t2Fmap3092691C67FB3059308B",
.6        FSequence' Fmap $\mathbb{Z}, \mathbb{Z}$ ] ( $\lambda x : \mathbb{Z} \cdot x \bmod 3$ ) ([1, 2, 3, 4, 5]) =
[1, 2, 0, 1, 2] ∧
.7        FSequence' Fmap[char*, char*]
.8        (
.9          FSequence' Take[char] (2)) (["Sahara", "Sakoh"]) =
["Sa", "Sa"]);

```

0.2.13 Index, IndexAll を検査する

```

63.0  t13 : () → FTestDriver' TestCase*
.1    t13 () ≜
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT13                                     :
\ t2Index, 2IndexAll3092691C67FB3059308B",
.6        let index = FSequence' Index,
.7          indexAll = FSequence' IndexAll in
.8          index[Z] (1) ([1, 2, 3, 4, 5]) = 1 ∧
.9          index[Z] (5) ([1, 2, 3, 4, 5]) = 5 ∧
.10         index[Z] (9) ([1, 2, 3, 4, 5]) = 0 ∧
.11         index[char] ('b') ('a', 'b', 'c') = 2 ∧
.12         index[char] ('z') ('a', 'b', 'c') = 0 ∧
.13         indexAll[Z] (9) ([1, 2, 3, 4, 5]) = {} ∧
.14         indexAll[Z] (9) ([]) = {} ∧
.15         indexAll[Z] (1) ([1, 2, 3, 4, 1]) = {1, 5} ∧
.16         indexAll[Z] (1) ([1, 2, 3, 4, 1, 1]) = {1, 5, 6}]);

```

0.2.14 Average を検査する

```

64.0  t14 : () → FTestDriver' TestCase*
.1    t14 () ≜
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT14                                     :
\ t2Average3092691C67FB3059308B",
.6        let avg1 = FSequence' Average[Z],
.7          avg2 = FSequence' Average[R] in
.8          avg1 ([]) = nil ∧
.9          avg1 ([1, 2, 3, 4]) = (1 + 2 + 3 + 4)/4 ∧
.10         avg2 ([1.3, 2.4, 3.5]) = 7.2/3)];

```

0.2.15 挿入を検査する

```

65.0  t15 : () → FTestDriver' TestCase*
.1    t15 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT15" :
\2633F51653092691C67FB3059308B",
.6        let ins1 = FSequence'InsertAt[ℤ],
.7          ins2 = FSequence'InsertAt[char] in
.8          ins1 (1) (1) ([2, 3, 4, 5]) = [1, 2, 3, 4, 5] ∧
.9          ins1 (3) (3) ([1, 2, 4, 5]) = [1, 2, 3, 4, 5] ∧
.10         ins1 (3) (3) ([1, 2]) = [1, 2, 3] ∧
.11         ins1 (4) (3) ([1, 2]) = [1, 2, 3] ∧
.12         ins1 (5) (3) ([1, 2]) = [1, 2, 3] ∧
.13         ins2 (1) ('1') ("2345") = "12345" ∧
.14         ins2 (3) ('3') ("1245") = "12345" ∧
.15         ins2 (3) ('3') ("12") = "123"]];

```

0.2.16 削除を検査する

```

66.0  t16 : () → FTestDriver' TestCase*
.1    t16 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT16" :
\2524A96643092691C67FB3059308B",
.6        let rm1 = FSequence'RemoveAt[ℤ],
.7          rm2 = FSequence'RemoveAt[char] in
.8          rm1 (1) ([1, 2, 3, 4, 5]) = [2, 3, 4, 5] ∧
.9          rm1 (3) ([1, 2, 4, 3]) = [1, 2, 3] ∧
.10         rm1 (3) ([1, 2]) = [1, 2] ∧
.11         rm1 (4) ([1, 2]) = [1, 2] ∧
.12         rm1 (5) ([1, 2]) = [1, 2] ∧
.13         rm2 (1) ("12345") = "2345" ∧
.14         rm2 (3) ("1243") = "123" ∧
.15         rm2 (3) ("12") = "12"]];

```

0.2.17 更新を検査する

```

67.0  t17 : () → FTestDriver' TestCase*
.1    t17 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT17
\266F465B03092691C67FB3059308B",
.6        let up1 = FSequence' UpdateAt[ℤ],
.7          up2 = FSequence' UpdateAt[char] in
.8          up1 (1) (10) ([1, 2, 3, 4, 5]) = [10, 2, 3, 4, 5] ∧
.9          up1 (3) (40) ([1, 2, 4, 3]) = [1, 2, 40, 3] ∧
.10         up1 (2) (30) ([1, 2]) = [1, 30] ∧
.11         up1 (3) (30) ([1, 2]) = [1, 2] ∧
.12         up1 (4) (30) ([1, 2]) = [1, 2] ∧
.13         up2 (1) ('a') ("12345") = "a2345" ∧
.14         up2 (3) ('b') ("1243") = "12b3" ∧
.15         up2 (3) ('c') ("123") = "12c" ∧
.16         up2 (3) ('c') ("12") = "12"]];

```

0.2.18 複数削除を検査する


```

68.0  t18 : () → FTestDriver' TestCase*
.1    t18 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT18                                     :
\289076570524A96643092691C67FB3059308B",
.6        let removeDup = FSequence' RemoveDup[ℤ],
.7          removeMember = FSequence' RemoveMember[ℤ],
.8          removeMembers = FSequence' RemoveMembers[ℤ] in
.9          removeDup ([]) = [] ∧
.10         removeDup ([1, 1, 2, 2, 2, 3, 4, 4, 4, 4]) = [1, 2, 3, 4] ∧
.11         removeDup ([1, 2, 3, 4]) = [1, 2, 3, 4] ∧
.12         removeMember (1) ([]) = [] ∧
.13         removeMember (1) ([1, 2, 3]) = [2, 3] ∧
.14         removeMember (4) ([1, 2, 3]) = [1, 2, 3] ∧
.15         removeMembers ([]) ([]) = [] ∧
.16         removeMembers ([]) ([1, 2, 3]) = [1, 2, 3] ∧
.17         removeMembers ([1, 2, 3]) ([]) = [] ∧
.18         removeMembers ([1, 2, 3]) ([1, 2, 3]) = [] ∧
.19         removeMembers ([1, 4, 5]) ([1, 2, 3, 4]) = [2, 3] ∧
.20         removeMembers ([1, 4, 5]) ([1, 2, 3, 4, 1, 2, 3, 4, 1]) =
[2, 3, 1, 2, 3, 4, 1]);

```

0.2.19 zip を検査する

```

69.0  t19 : () → FTestDriver' TestCase*
.1    t19 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT19 : \t2zip3092691C67FB3059308B",
.6        let zip = FSequence' Zip[ℤ, char],
.7        zip2 = FSequence' Zip2[ℤ, char],
.8        unzip = FSequence' Unzip[ℤ, char] in
.9        zip ([], []) = [] ∧
.10       zip ([1, 2, 3], ['a', 'b', 'c']) =
[mk- (1, 'a'), mk- (2, 'b'), mk- (3, 'c')] ∧
.11       zip ([1, 2], ['a', 'b', 'c']) = [mk- (1, 'a'), mk- (2, 'b')] ∧
.12       zip ([1, 2, 3], ['a', 'b']) = [mk- (1, 'a'), mk- (2, 'b')] ∧
.13       zip2 ([[]] ([[]]) = [] ∧
.14       zip2 ([1, 2, 3]) (['a', 'b', 'c']) =
[mk- (1, 'a'), mk- (2, 'b'), mk- (3, 'c')] ∧
.15       unzip ([[]]) = mk- ([[], []]) ∧
.16       unzip ([mk- (1, 'a'), mk- (2, 'b'), mk- (3, 'c')]) =
mk- ([1, 2, 3], ['a', 'b', 'c']));

```

0.2.20 Span を検査する

```

70.0  t20 : () → FTestDriver' TestCase*
.1    t20 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT20 : \t2Span3092691C67FB3059308B",
.6        let span = FSequence' Span[ℤ],
.7        p1 = λ x : ℤ · x mod 2 = 0,
.8        p2 = λ x : ℤ · x < 10 in
.9        span (p1) ([[]]) = mk- ([[], []]) ∧
.10       span (p1) ([2, 4, 6, 1, 3]) = mk- ([2, 4, 6], [1, 3]) ∧
.11       span (p2) ([1, 2, 3, 4, 5]) = mk- ([1, 2, 3, 4, 5], []) ∧
.12       span (p2) ([1, 2, 12, 13, 4, 15]) =
mk- ([1, 2], [12, 13, 4, 15]));

```

0.2.21 TakeWhile, DropWhile を検査する

```

71.0  t21 : () → FTestDriver' TestCase*
.1    t21 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT21
\ t2TakeWhile, 2DropWhile3092691C67FB3059308B",
.6        let TakeWhile = FSequence' TakeWhile[ℤ],
.7            DropWhile = FSequence' DropWhile[ℤ],
.8            p1 = λ x : ℤ · x mod 2 = 0 in
.9            TakeWhile (p1) ([ ]) = [ ] ∧
.10           TakeWhile (p1) ([2, 4, 6, 8, 1, 3, 5, 2, 4]) = [2, 4, 6, 8] ∧
.11           DropWhile (p1) ([ ]) = [ ] ∧
.12           DropWhile (p1) ([2, 4, 6, 8, 1, 2, 3, 4, 5]) = [1, 2, 3, 4, 5]);

```

0.2.22 Foldl を検査する

```

72.0  t22 : () → FTestDriver' TestCase*
.1    t22 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT22 : \ t2Foldl3092691C67FB3059308B",
.6        let foldl = FSequence' Foldl[ℤ, ℤ],
.7            f2 = FSequence' Foldl[char*, char],
.8            plus = FSequence' Plus[ℤ],
.9            prod = FSequence' Product[ℤ] in
.10           foldl (plus) (0) ([1, 2, 3]) = 6 ∧
.11           foldl (prod) (1) ([2, 3, 4]) = 24 ∧
.12           f2 (FSequence' Append[char]) ([ ]) ("abc") = "abc");

```

0.2.23 Foldr を検査する

```

73.0  t23 : () → FTestDriver' TestCase*
.1    t23 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT23 : \t2Foldr3092691C67FB3059308B",
.6        let removeAt = FSequence' RemoveAt[char],
.7          foldr = FSequence' Foldr[Z, Z],
.8          f3 = FSequence' Foldr[N1, char*],
.9          plus = FSequence' Plus[Z],
.10         prod = FSequence' Product[Z] in
.11         foldr (plus) (0) ([1, 2, 3]) = 6 ∧
.12         foldr (prod) (1) ([2, 3, 4]) = 24 ∧
.13         f3 (removeAt) ("12345") ([1, 3, 5]) = "24");

```

0.2.24 IsDup を検査する

```

74.0  t24 : () → FTestDriver' TestCase*
.1    t24 () △
.2    [
.3      mk-FTestDriver' TestCase
.4      (
.5        "FSequenceT24 : \t2IsDup3092691C67FB3059308B",
.6        FSequence' IsDup[Z] ([1, 2, 3]) = false ∧
.7        FSequence' IsDup[Z] ([1, 2, 2, 3]) ∧
.8        FSequence' IsDup[Z] ([1]) = false ∧
.9        FSequence' IsDup[Z] ([1, 2, 3, 1]))
end FSequenceT

```