VDM++ 関数型ライブラリのドキュメント雛形

佐原　伸

(株)CSK

目次

1 はじめに

本ドキュメントは、VDM++ 関数型ライブラリ・ドキュメントの雛形であり、まだ、すべての VDM モジュールを記述しているわけではない。

2 関数型ライブラリのドキュメント

2.1 AllT

すべてのテストケースを実行し、結果列を返す。

```
class
AllT
values
  lg = new FTestLogger ()
functions
public static
  run : () -> bool
  run () ==
    let testcases = [
                    FSequenceT`run (),
                    FSetT`run (),
                    FCharT`run (),
                    FRealT`run (),
                    FFunctionT`run (),
                    FMapT`run (),
                    FHashtableT`run (),
                    FIntegerT`run (),
                    FNumberT`run (),
                    FProductT`run (),
                    FQueueT`run (),
                    FCalendarT`run (),
                    FBusinessTableT`run (),
                    FJapaneseCalendarT`run ()] in
    if  forall i in set inds testcases & testcases (i)
    then lg.Print ("全テストケースが OK")
    else lg.Print ("誤りがあります。上の各テスト結果で NG が無いかチェックしてください。")
end
AllT
```

2.2 FBusinessTable

ビジネス上の規則に使用される表に関わる関数を提供する。

```
class
FBusinessTable
values
public
  MAXNUMBER = 2 ** 52
```

Entry は、lower ¡= key ¡= upper であれば、data が対応することを示す表の要素である。表 Table は Entry の列で表す。

```
types
public
  Entry::lower : rat
         upper : rat
         data : rat;
  public EntryTable = seq of Entry
  inv wEntryTable ==
       forall i,j in set inds wEntryTable &
           下限より上限が大きい(wEntryTable(i).lower, wEntryTable(i).upper) and
           j = i + 1 =>
           上限と次の行の下限は等しい(wEntryTable(i).upper, wEntryTable(j).lower)
```

Find は、表 table を検索して、key に対応する data を返す。対応する値がない場合は nil を返す。

```
functions
public static
  Find : EntryTable * rat +> [rat]
  Find(table,key) ==
    if  exists1 i in set inds table & table(i).lower < key and key <= table(i).upper
    then let  n in set inds tablebe st  table(n).lower < key and key <= table(n).upper in
         table(n).data
    else nil
  post if  exists1 i in set inds table & table(i).lower < key and key <= table(i).upper
       then let  n in set inds tablebe st  table(n).lower < key and key <= table(n).upper in
            RESULT = table(n).data
       else RESULT = nil ;
```

FindInRegularIntervalsTables は、一定間隔 (interval) の unit を持つ表 table を検索して、key に対応する unit 数を返す。例えば、「売買代金 1,000 万円超の場合は、100 万円まで毎に 50 円追加」という規則があった場合に、売買代金 key=11300000 の追加料金計算をするには、FindInRegularIntervalsTables(10000000, 50, 1000000, 11300000) とすれば 100 を返す。key ¡ base ならば、0 を返す。

```
public static
```

```
  FindInRegularIntervalsTables : int * int * int * rat +> rat
  FindInRegularIntervalsTables (base,unit,interval,key) ==
    let x = key-base,
        q = x div interval,
        m =
            if (x mod interval) = 0
            then 0
            else 1 in
    if x < 0
    then 0
    else (q + m) * unit;
public static
  下限より上限が大きい : rat * rat +> bool
  下限より上限が大きい (a 下限,a 上限) ==
    a 下限 < a 上限;
public static
  上限と次の行の下限は等しい : rat * rat +> bool
  上限と次の行の下限は等しい (a 下限,a 上限) ==
    a 下限 = a 上限
end
FBusinessTable
```

Test Suite : vdm.tc

Class : FBusinessTable

Name	#Calls	Coverage
FBusinessTable‘Find	17	√
FBusinessTable‘下限より上限が大きい	475	√
FBusinessTable‘上限と次の行の下限は等しい	76	√
FBusinessTable‘FindInRegularIntervalsTables	10	√
Total Coverage		**100%**

2.3 FBusinessTableT

FBusinessTable のテストを行う。

```
class
FBusinessTableT is subclass of FBusinessTable
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 ()] in
    FTestDriver`run (testcases);
```

2.3.1 Find を検査する

```
t1 : () +> FTestDriver`TestCase
t1() ==
  mk_FTestDriver`TestCase
      (
       "FBusinessTableT01 : \t Find",
       let t : EntryTable = [
                            mk_Entry(0, 3000000, 3000),
                            mk_Entry(3000000, 6000000, 6000),
                            mk_Entry(6000000, 9000000, 9000),
                            mk_Entry(9000000, 12000000, 12000),
                            mk_Entry(12000000, MAXNUMBER, 15000)] in
       Find(t, -1) = nil and
       Find(t, 0) = nil and
       Find(t, 1000) = 3000 and
       Find(t, 3000000) = 3000 and
       Find(t, 3000001) = 6000 and
       Find(t, 3000002) = 6000 and
       Find(t, 6000000) = 6000 and
       Find(t, 6000001) = 9000 and
       Find(t, 6000002) = 9000 and
       Find(t, 9000000) = 9000 and
       Find(t, 9000001) = 12000 and
       Find(t, 9000002) = 12000 and
       Find(t, 12000000) = 12000 and
       Find(t, 12000001) = 15000 and
       Find(t, 12000002) = 15000 and
       Find(t, MAXNUMBER) = 15000 and
       Find(t, MAXNUMBER + 1) = nil);
```

2.3.2 FindInRegularIntervalsTables を検査する

```
  t2 : () +> FTestDriver`TestCase
  t2() ==
    mk_FTestDriver`TestCase
        (
         "FBusinessTableT02 : \t FindInRegularIntervalsTables",
         FindInRegularIntervalsTables(10000000, 50, 1000000, 0) = 0 and
         FindInRegularIntervalsTables(10000000, 50, 1000000, 9999999) = 0 and
         FindInRegularIntervalsTables(10000000, 50, 1000000, 10000000) = 0 and
         FindInRegularIntervalsTables(10000000, 50, 1000000, 10300000) = 50 and
         FindInRegularIntervalsTables(10000000, 50, 1000000, 11000000) = 50 and
         FindInRegularIntervalsTables(10000000, 50, 1000000, 11300000) = 100 and
         FindInRegularIntervalsTables(0, 3000, 3000000, 0) = 0 and
         FindInRegularIntervalsTables(0, 3000, 3000000, 1) = 3000 and
         FindInRegularIntervalsTables(0, 3000, 3000000, 3000000) = 3000 and
         FindInRegularIntervalsTables(0, 3000, 3000000, 3000001) = 6000)
end
FBusinessTableT
```

2.4 FCalendar

グレゴリオ暦に関わる関数を定義する。

```
class
FCalendar
```

使用する型は以下の通りである。Date は、修正ユリウス日を表すので実数型である。DayOfWeek は、曜日計算が便利なように 0,...,6 の値を各曜日にふっていて、日曜日が 0、土曜日が 6 である。

```
types
  public Date = real;
  public DayOfWeekName =  <Sun> | <Mon> | <Tue> | <Wed> | <Thu> | <Fri> | <Sat> ;
  public DayOfWeek = nat
  inv dayOfWeek == dayOfWeek <= 6
```

使用する値は以下の通り。diffJDandMJD はユリウス日 (Julian Date) と修正ユリウス日 (Modified Julian Date) の日数差である。reviseMonths は、日付計算が便利なように使用する月数を意味する。

```
values
private
  diffJDandMJD = 2400000.5;
private
  DayOfWeekSequence = [ <Sun> , <Mon> , <Tue> , <Wed> , <Thu> , <Fri> , <Sat> ];
private
  daysInYear = 365.25;
private
  monthsInYear = 12;
private
  reviseMonths = 14;
private
  daysInWeek = 7;
private
  averageDaysInMonth = 30.6001;
private
  yearsInCentury = 100;
private
  constForDayCalc = 122.1;
private
  constForYearCalc = 4800;
private
  constForCenturyCalc = 32044.9;
private
  theDayBeforeGregorianCalendar = 2299160;
```

```
private
  theFirstDayOfGregorianCalendar = 1582.78;
public
  max = FNumber‘Max[real](FNumber‘GT);
public
  min = FNumber‘Min[real](FNumber‘GT)
```

DateFromInt は、(yyyy)(mm)(dd) で表した暦日付に対応する日付を返す。日 dd は、(1,...,31) でなくてよい。0 日は、前月末尾に補正され、12 月 32 日は翌年 1 月 1 日に補正される。

```
functions
public static
  DateFromInt : int +> int +> rat +> Date
  DateFromInt(yyyy)(mm)(dd) ==
    let [year, month] =
            if (mm > reviseMonths-monthsInYear)
            then [yyyy + constForYearCalc, mm + 1]
            else [yyyy + constForYearCalc-1, mm + reviseMonths-1],
        aCentury = year div yearsInCentury,
        constCentury =
            if (ConvToYear(yyyy, mm, dd) > theFirstDayOfGregorianCalendar)
            then aCentury div 4-aCentury-32167
            else -32205,
        halfDay = 0.5 in
    floor (daysInYear * year) +
    floor (averageDaysInMonth * month) +
    dd +
    constCentury-halfDay-diffJDandMJD;
```

GetLegalDate は、年月日を通常の値の範囲内に変換する。月 tempM は、(1,...12) でなく、13 以上や 0 や負数でもよい。例えば、13 は翌年 1 月に補正され、0 年は前年 12 月に補正される。日 dd も、(1,...,31) でなくてよい。0 日は、前月末尾に補正され、12 月 32 日は翌年 1 月 1 日に補正される。

```
public static
  GetLegalDate : int +> int +> int +> Date
  GetLegalDate(tmpY)(tempM)(dd) ==
    let mk_(yyyy, mm) = GetLegalMonth(tmpY)(tempM) in
    DateFromInt(yyyy)(mm)(dd);
```

GetLegalMonth は、年月を通常の値の範囲内に変換する。

```
public static
```

```
  GetLegalMonth : int +> int +> int * int
  GetLegalMonth (tempY)(tempM) ==
    let y =
            if tempM <= 0
            then tempY + (tempM-12) div monthsInYear
            else tempY + (tempM-1) div monthsInYear,
        m = FInteger`amod (tempM) (monthsInYear) in
    mk_(y,m);
```

Int3FromDate は、日付から三つ組み数 (yyyy, mm, dd) で表した暦日付を得る。

```
public static
  Int3FromDate : Date +> int * int * int
  Int3FromDate (aDate) ==
    mk_(Year (aDate),Month (aDate),Day (aDate));
```

Year は、日付から、その日付の属する年を得る。

```
public static
  Year : Date +> int
  Year (aDate) ==
    if MonthAux (aDate) < reviseMonths
    then YearAux (aDate)-constForYearCalc
    else YearAux (aDate)-constForYearCalc + 1;
```

Month は、日付から、その日付の属する月を得る。

```
public static
  Month : Date +> int
  Month (aDate) ==
    if MonthAux (aDate) < reviseMonths
    then MonthAux (aDate)-1
    else MonthAux (aDate)-13;
```

Day は、日付から、日を得る。

```
public static
  Day : Date +> int
  Day (aDate) ==
    DayInMonth (aDate);
```

DayInYear は、年日付を得る。

```
public static
  DayInYear : Date +> int
  DayInYear (aDate) ==
    let firstDay = DateFromInt (Year (aDate)) (1) (0) in
    Diff (aDate) (firstDay);
```

DayInMonth は、月日付けを得る。

```
public static
  DayInMonth : Date +> int
  DayInMonth (aDate) ==
    floor (DayInMonthAsReal (aDate));
```

DayInMonthAsReal は、実数の月日付けを得る。

```
  DayInMonthAsReal : Date +> real
  DayInMonthAsReal (aDate) ==
    YMDAUX (aDate) + constForDayCalc-floor (daysInYear * YearAux (aDate))-
    floor (averageDaysInMonth * MonthAux (aDate));
```

MonthAux は、日付計算上都合の良い月 (4..15) を返す補助関数。

```
  MonthAux : Date +> int
  MonthAux (aDate) ==
    floor ((YMDAUX (aDate) + constForDayCalc-
      floor (daysInYear * YearAux (aDate)))/
     averageDaysInMonth);
```

YMDAUX は、日付を年月日に変更するための補助関数。グレゴリオ暦切替前と、グレゴリオ暦切替後の考慮を行っている。

```
  YMDAUX : Date +> real
  YMDAUX (aDate) ==
    let JD = MJD2JD (aDate),
        aCentury = floor ((JD + constForCenturyCalc)/36524.25) in
    if JD > theDayBeforeGregorianCalendar
    then JD + constForCenturyCalc + aCentury-aCentury div 4 + 0.5
    else JD + 32082.9 + 0.5;
```

YearAux は、日付から日付計算に都合の良い補正をした年数を求めるための補助関数。

```
  YearAux : Date +> int
  YearAux (aDate) ==
    floor (YMDAUX (aDate)/daysInYear);
```

ConvToYear は、（整数三つ組の）暦日付を年[*1]に変換する。例えば、ConvToYear(2001,7,1) は 2001.5 を返す。

```
public static
  ConvToYear : int * int * rat +> real
  ConvToYear (yyyy, mm, dd) ==
    yyyy + (mm-1)/monthsInYear + (floor (dd)-1)/daysInYear;
```

[*1] 整数部が年、小数点以下が年の中での日付を表す形式で、修正ユリウス日ではない。

MJD2JD は、修正ユリウス日をユリウス日に変換する。

```
public static
  MJD2JD : Date +> real
  MJD2JD (aMJD) ==
    aMJD + diffJDandMJD;
```

JD2MJD は、ユリウス日を修正ユリウス日すなわち日付に変換する。

```
public static
  JD2MJD : real +> Date
  JD2MJD (aJD) ==
    aJD-diffJDandMJD;
```

2.4.1 計算関数群

Diff は、 2 つの日付 d1, d2 の差を得る。

```
public static
  Diff : Date +> Date +> int
  Diff (d1)(d2) ==
    floor (d1-d2);
```

2.4.2 照会関数群

IsLeapYear は、閏年であれば true、平年であれば false を返す。

```
public static
  IsLeapYear : int +> bool
  IsLeapYear (yyyy) ==
    yyyy mod 400 = 0 or (yyyy mod yearsInCentury <> 0 and yyyy mod 4 = 0);
```

GetDayOfWeek は、曜日数を得る。

```
public static
  GetDayOfWeek : Date +> DayOfWeek
  GetDayOfWeek (d) ==
    (floor (d)-4) mod daysInWeek;
```

GetDayOfWeekName は、曜日名を得る。

```
public static
  GetDayOfWeekName : Date +> DayOfWeekName
  GetDayOfWeekName (d) ==
    DayOfWeekSequence (GetDayOfWeek (d) + 1);
```

GetDayOfWeekFromName は、曜日名から曜日数を求める。

```
public static
  GetDayOfWeekFromName : DayOfWeekName +> DayOfWeek
  GetDayOfWeekFromName (dn) ==
    FSequence`Index[DayOfWeekName] (dn) (DayOfWeekSequence)-1;
```

FirstDayOfWeekOfMonth は、指定した yyyy 年 m 月の最初の dayOfWeekName 曜日名の日付を得る。

```
public static
  FirstDayOfWeekOfMonth : DayOfWeekName +> int +> int +> Date
  FirstDayOfWeekOfMonth (dayOfWeekName)(m)(yyyy) ==
    let dayOfWeek = GetDayOfWeekFromName (dayOfWeekName),
        firstDayOfMonth = GetFirstDayOfMonth (m) (yyyy),
        diff = dayOfWeek-GetDayOfWeek (firstDayOfMonth) in
    cases true :
      (diff = 0) -> firstDayOfMonth,
      (diff > 0) -> firstDayOfMonth + diff,
      (diff < 0) -> firstDayOfMonth + ((daysInWeek + diff) mod daysInWeek)
    end;
```

GetLastDayOfWeekOfMonth は、指定した yyyy 年 m 月の最後の dayOfWeekName 曜日名の日付を得る。

指定された月の翌月の最初の指定曜日から 7 日前を返す。月が 1 2 月の場合でも本クラスの関数は yyyy 年 13 月を yyyy+1 年 1 月と解釈するので、問題ない。

```
public static
  GetLastDayOfWeekOfMonth : DayOfWeekName +> int +> int +> Date
  GetLastDayOfWeekOfMonth (dayOfWeekName)(m)(yyyy) ==
    FirstDayOfWeekOfMonth (dayOfWeekName) (m + 1) (yyyy)-daysInWeek;
```

GetNthDayOfWeekOfMonth は、指定された yyyy 年 m 月 dayOfWeekName 曜日名の、第 n 曜日を求める。第 n 曜日が存在しなければ nil を返す。

月初指定曜日の (n - 1) * 7 日後を返す。

```
public static
  GetNthDayOfWeekOfMonth : DayOfWeekName +> int +> int +> int +> [Date]
  GetNthDayOfWeekOfMonth (dayOfWeekName)(n)(m)(yyyy) ==
    let firstDayOfWeekOfMonth = FirstDayOfWeekOfMonth (dayOfWeekName) (m) (yyyy),
        r = firstDayOfWeekOfMonth + (daysInWeek * (n-1)) in
    cases Month (r) :
      (m) -> r,
      others -> nil
    end;
```

GetFirstDayOfMonth は、指定した yyyy 年 m 月の月初日を得る。

```
public static
  GetFirstDayOfMonth : int +> int +> Date
  GetFirstDayOfMonth (m)(yyyy) ==
    GetLegalDate (yyyy) (m) (1);
```

GetLastDayOfMonth は、指定した yyyy 年 m 月の月末日を求める。

翌月の月初日の 1 日前を返す。

```
public static
```

```
  GetLastDayOfMonth : int +> int +> Date
  GetLastDayOfMonth (m)(yyyy) ==
    GetLegalDate (yyyy) (m + 1) (1)-1;
```

IsSunday は、指定日が日曜日か否かを返す。

```
public static
  IsSunday : Date +> bool
  IsSunday (d) ==
    GetDayOfWeek (d) = 0;
```

IsSaturday は、指定日が土曜日か否かを返す。

```
public static
  IsSaturday : Date +> bool
  IsSaturday (d) ==
    GetDayOfWeek (d) = 6;
```

IsWeekDay は、指定日がウィークデイか否かを返す。

```
public static
  IsWeekDay : Date +> bool
  IsWeekDay (d) ==
    GetDayOfWeek (d) in set {1,...,5};
```

IsDayOFWeekNameWeekDay は、指定した dayOfWeekName 曜日名がウィークデイか否かを返す。

```
public static
  IsDayOFWeekNameWeekDay : DayOfWeekName +> bool
  IsDayOFWeekNameWeekDay (dayOfWeekName) ==
    dayOfWeekName not in set { <Sat> , <Sun> };
```

2.4.3 指定された曜日が何日あるかを返す関数

HowManyDayOfWeekWithin2Days は、指定された dayOfWeekName 曜日名が、指定された日付間（d1 と d2 の間）に何日あるかを返す。d1 と d2 が指定された曜日であれば勘定に入れる。

以下は、HowManyDayOfWeekWithin2Days 関数の山崎利治さんによる段階的洗練を佐原が「翻訳」した記述である。

前件は以下である。

$$\texttt{type R} = \{|\texttt{rng[n} \to \texttt{n/7|n} \in \texttt{Int]}|\}\ \text{（注）}7\ \text{で割った商の集合}$$

$$\texttt{f,t} \in \texttt{Int,w} \in \texttt{R,0} \leq \texttt{f} \leq \texttt{t,h : Int} \to \texttt{R}\ \text{（注）環準同型（}\textit{ring homomorphism}\text{）}$$

後件は以下のようになる。

$$\texttt{exists S \& } \in \texttt{Int} \bullet \texttt{S} = \texttt{h}^{-1}\texttt{(w)} \cap \texttt{f..t} \wedge \text{答え} \equiv \texttt{card(S)}$$

すなわち、整数系を環（ring）と見て、その商環（quotient ring）への準同型写像があり、その代数系上で後件（事後条件）を満たすプログラムを作るという問題に抽象化された。HowManyDayOfWeekWithin2Days は、**7 で割る**特殊な場合の実装であるということになる。

```
I = {f..t}
d = t - f + 1  -- = card(I)
q = d / 7
r = d \ 7  --7 で割った余り
```

とすると、答え A に対して q ≤ A ≤ q + 1 が成り立つ。なぜなら、

- 任意の連続する 7 日間には、必ず w 曜日がちょうど 1 日存在する。
- card(I) = 7 × q + r(0 ≤ r<7) であるから、I には少なくとも q 個の連続する 7 日間が存在するが、q+1 個は存在しない。
- 余りの r 日間に w 曜日が存在するかも知れない。

次に、

```
x ++ y = (x + y) \ 7
x ⊥ y = max(x - y, 0)
```

として、

```
T = {h(f)..h(f) ++ (r ⊥ 1)}
```

を考える。T は余り r 日間の曜日に対応する（card(T) = r)。すると、

```
A ≡ if w ∈ T then q + 1 els q end
```

ここで、

```
x minus y = if x ≧ y then x - y else x - y + 7 end
```

とすれば、

```
w ∈ T ⇔ (w minus h(f)) + 1 ≦ r
```

である。なぜならば

```
w ∈ T ⇔ {0..(r ⊥ 1)}∋ w^^ef^^be^^95 = w minus h(f)
⇔ r ⊥ 1 ≧ w^^ef^^be^^95
⇔ r ≧ (w minus h(f)) + 1
```

従って、プログラムは以下のようになる。

```
A(f, t w) ≡
let
d ≡ t - f + 1
q ≡ d / 7
r ≡ d \ 7
delta ≡ if (w minus h(f)) + 1 ≦ r then 1 els 0 end
x minus y ≡ if x ≧ y then x - y els x - y + 7 end
```

```
in
q + delta
end
```

あとは、上記プログラムを VDM++ に翻訳すればよい。

```
public static
  HowManyDayOfWeekWithin2Days : DayOfWeekName +> Date +> Date +> int
  HowManyDayOfWeekWithin2Days (dayOfWeekName)(d1)(d2) ==
    let dayOfWeek = GetDayOfWeekFromName (dayOfWeekName),
        f = min (d1) (d2),
        t = max (d1) (d2),
        days = Diff (t) (f) + 1,
        q = days div daysInWeek,
        r = days mod daysInWeek,
        delta = if SubtractDayOfWeek (dayOfWeek) (GetDayOfWeek (f)) + 1 <= r
                then 1
                else 0 in
    q + delta;
```

SubtractDayOfWeek は、曜日数の減算を行う。

```
private static
  SubtractDayOfWeek : int +> int +> int
  SubtractDayOfWeek (x)(y) ==
    if x >= y
    then x-y
    else x-y + daysInWeek;
```

GetVernalEquinoxInGMT は、yyyy 年のグリニッジ標準時の春分を得る。

```
public static
  GetVernalEquinoxInGMT : int +> Date
  GetVernalEquinoxInGMT (yyyy) ==
    let y = yyyy/1000 in
    JD2MJD (1721139.2855 + 365.242138 * yyyy + y * y * (0.067919-0.002788 * y));
```

GetSummerSolsticeInGMT は、yyyy 年のグリニッジ標準時の夏至を得る。

```
public static
  GetSummerSolsticeInGMT : int +> Date
  GetSummerSolsticeInGMT (yyyy) ==
    let y = yyyy/1000 in
    JD2MJD (1721233.2486 + 365.241728 * yyyy-y * y * (0.053018-0.009332 * y));
```

GetAutumnalEquinoxInGMT は、yyyy 年のグリニッジ標準時の秋分を得る。

```
public static
```

```
  GetAutumnalEquinoxInGMT : int +> Date
  GetAutumnalEquinoxInGMT (yyyy) ==
    let y = yyyy/1000 in
    JD2MJD (1721325.6978 + 365.242505 * yyyy-y * y * (0.126689-0.00194 * y));
```

GetWinterSolsticeInGMT は、yyyy 年のグリニッジ標準時の冬至を得る。

```
public static
  GetWinterSolsticeInGMT : int +> Date
  GetWinterSolsticeInGMT (yyyy) ==
    let y = yyyy/1000 in
    JD2MJD (1721414.392 + 365.24289 * yyyy-y * y * (0.010965-0.008485 * y));
```

GetDateInST は、GMT 基準の日付と、求めたい（日本標準時などの）標準時との差 diff（単位=時間）を与えて、標準時基準の日付を得る。日本の場合、日本標準時 = GMT + 9 時間。

```
public static
  GetDateInST : rat +> Date +> Date
  GetDateInST (diff)(d) ==
    floor (d + diff/24);
```

2.4.4 休日に関わる照会関数群

以下は、休日の考慮をした機能である。

GetHolidaysWithinDates は、ある年の休日の集合を得る getHolidays 関数で決まる、2 つの日付の間の休日の集合を返す。日曜日である休日も含むが、休日でない日曜日は含まない。

```
public static
  GetHolidaysWithinDates : (int +> set of Date) +> Date +> Date +> set of Date
  GetHolidaysWithinDates (getHolidays)(d1)(d2) ==
    let date1 = min (d1) (d2),
        date2 = max (d1) (d2),
        setOfYear = {Year (date1),...,Year (date2)},
        holidays = dunion {getHolidays (y)|y in set setOfYear} in
    {h|h in set holidays & d1 <= h and h <= d2};
```

GetHolidaysWithinDatesNotSunday は、ある年の休日の集合を得る getHolidays 関数で決まる、2 つの日付の間の日曜日を含まない休日の集合を返す。

```
public static
  GetHolidaysWithinDatesNotSunday : (int +> set of Date) +> Date +> Date +> set of Date
  GetHolidaysWithinDatesNotSunday (getHolidays)(d1)(d2) ==
    let holidays = GetHolidaysWithinDates (getHolidays) (d1) (d2) in
    {h|h in set holidays & not IsSunday (h)};
```

GetHolidaysWithinDatesAsSunday は、日曜日である休日の集合を返す。

```
public static
```

```
  GetHolidaysWithinDatesAsSunday : (int +> set of Date) +> Date +> Date +> set of Date
  GetHolidaysWithinDatesAsSunday (getHolidays)(d1)(d2) ==
    let holidays = GetHolidaysWithinDates (getHolidays) (d1) (d2) in
    {h|h in set holidays & IsSunday (h)};
```

GetNumberOfHolidaysWithinDates は、２つの日付の間の休日数を返す。日曜日である休日も含むが、休日でない日曜日は含まない。

```
public static
  GetNumberOfHolidaysWithinDates : (int +> set of Date) +> Date +> Date +> int
  GetNumberOfHolidaysWithinDates (getHolidays)(d1)(d2) ==
    card GetHolidaysWithinDates (getHolidays) (d1) (d2);
```

GetNumberOfDayOff は、２つの日付の間の休日あるいは日曜日の数を返す（両端が該当日であれば含む）

```
public static
  GetNumberOfDayOff : (int +> set of Date) +> Date +> Date +> int
  GetNumberOfDayOff (getHolidays)(d1)(d2) ==
    let date1 = min (d1) (d2),
        date2 = max (d1) (d2),
        numOfSunday = HowManyDayOfWeekWithin2Days ( <Sun> ) (d1) (d2) in
    numOfSunday + card GetHolidaysWithinDatesNotSunday (getHolidays) (date1) (date2);
```

GetNumberOfDayOff1 は、２つの日付の間の休日あるいは日曜日の数を返す（開始日を含まない）

```
public static
  GetNumberOfDayOff1 : (int +> set of Date) +> Date +> Date +> int
  GetNumberOfDayOff1 (getHolidays)(d1)(d2) ==
    let date1 = min (d1) (d2),
        date2 = max (d1) (d2) in
    GetNumberOfDayOff (getHolidays) (date1 + 1) (date2);
```

2.4.5 休日に関わる計算関数群

BusinessDateToFuture は、休日でない日付を返す（未来へ向かって探索する）。

```
public static
  BusinessDateToFuture : (int +> set of Date) +> Date +> Date
  BusinessDateToFuture (getHolidays)(d) ==
    cases IsDayOff (getHolidays) (d) or IsSaturday (d) :
      true -> BusinessDateToFuture (getHolidays) (d + 1),
      others -> d
    end;
```

BusinessDateToPast は、休日でない日付を返す（過去へ向かって探索する）。

```
public static
```

```
  BusinessDateToPast : (int +> set of Date) +> Date +> Date
  BusinessDateToPast (getHolidays)(d) ==
    cases IsDayOff (getHolidays) (d) or IsSaturday (d) :
      true -> BusinessDateToPast (getHolidays) (d-1),
      others -> d
    end;
```

AddBusinessDays は、与えられた平日 d に、平日 n 日分を加算する。

```
public static
  AddBusinessDays : (int +> set of Date) +> Date +> int +> Date
  AddBusinessDays (getHolidays)(d)(n) ==
    AddBusinessDaysAux (getHolidays) (BusinessDateToFuture (getHolidays) (d)) (n);
public static
  AddBusinessDaysAux : (int +> set of Date) +> Date +> int +> Date
  AddBusinessDaysAux (getHolidays)(d)(n) ==
    cases IsDayOff (getHolidays) (d) or IsSaturday (d) :
      true -> AddBusinessDaysAux (getHolidays) (d + 1) (n),
      others -> if n <= 0
                then d
                else AddBusinessDaysAux (getHolidays) (d + 1) (n-1)
    end;
```

SubtractBusinessDays は、与えられた平日に、平日 n 日分を減算する。

```
public static
  SubtractBusinessDays : (int +> set of Date) +> Date +> int +> Date
  SubtractBusinessDays (getHolidays)(d)(n) ==
    SubtractBusinessDaysAux (getHolidays) (BusinessDateToPast (getHolidays) (d)) (n);
public static
  SubtractBusinessDaysAux : (int +> set of Date) +> Date +> int +> Date
  SubtractBusinessDaysAux (getHolidays)(d)(n) ==
    cases IsDayOff (getHolidays) (d) or IsSaturday (d) :
      true -> SubtractBusinessDaysAux (getHolidays) (d-1) (n),
      others -> if n <= 0
                then d
                else SubtractBusinessDaysAux (getHolidays) (d-1) (n-1)
    end;
```

2.4.6 休日に関わる検査関数群

IsHoliday は、指定した日 d が休日か否かを返す。

```
public static
```

```
  IsHoliday : (int +> set of Date) +> Date +> bool
  IsHoliday (getHolidays)(d) ==
    d in set getHolidays (Year (d));
```

IsDayOff は、指定した日 d が休み（休日または日曜日）であるか否かを返す。

```
public static
  IsDayOff : (int +> set of Date) +> Date +> bool
  IsDayOff (getHolidays)(d) ==
    IsSunday (d) or IsHoliday (getHolidays) (d)
end
FCalendar
```

Test Suite : vdm.tc

Class : FCalendar

Name	#Calls	Coverage
FCalendar‘Day	5	√
FCalendar‘Diff	36	√
FCalendar‘Year	197	√
FCalendar‘Month	589	√
FCalendar‘JD2MJD	313	√
FCalendar‘MJD2JD	2968	√
FCalendar‘YMDAUX	2967	√
FCalendar‘YearAux	1582	√
FCalendar‘IsDayOff	109	√
FCalendar‘IsSunday	2340	√
FCalendar‘MonthAux	1380	√
FCalendar‘DayInYear	6	√
FCalendar‘IsHoliday	100	√
FCalendar‘IsWeekDay	7	√
FCalendar‘ConvToYear	2329	√
FCalendar‘DayInMonth	5	√
FCalendar‘IsLeapYear	5	√
FCalendar‘IsSaturday	58	√
FCalendar‘DateFromInt	2329	√
FCalendar‘GetDateInST	306	√
FCalendar‘GetDayOfWeek	3053	√
FCalendar‘GetLegalDate	653	√
FCalendar‘Int3FromDate	4	√
FCalendar‘GetLegalMonth	675	√
FCalendar‘AddBusinessDays	4	√

Name	#Calls	Coverage
FCalendar‘DayInMonthAsReal	5	√
FCalendar‘GetDayOfWeekName	11	√
FCalendar‘GetLastDayOfMonth	31	√
FCalendar‘GetNumberOfDayOff	18	√
FCalendar‘SubtractDayOfWeek	29	√
FCalendar‘AddBusinessDaysAux	19	√
FCalendar‘BusinessDateToPast	22	√
FCalendar‘GetFirstDayOfMonth	604	√
FCalendar‘GetNumberOfDayOff1	9	√
FCalendar‘BusinessDateToFuture	31	√
FCalendar‘GetDayOfWeekFromName	636	√
FCalendar‘SubtractBusinessDays	5	√
FCalendar‘FirstDayOfWeekOfMonth	600	√
FCalendar‘GetVernalEquinoxInGMT	148	√
FCalendar‘GetHolidaysWithinDates	43	√
FCalendar‘GetNthDayOfWeekOfMonth	584	√
FCalendar‘GetSummerSolsticeInGMT	5	√
FCalendar‘GetWinterSolsticeInGMT	5	√
FCalendar‘IsDayOFWeekNameWeekDay	7	√
FCalendar‘GetAutumnalEquinoxInGMT	148	√
FCalendar‘GetLastDayOfWeekOfMonth	9	√
FCalendar‘SubtractBusinessDaysAux	31	√
FCalendar‘HowManyDayOfWeekWithin2Days	29	√
FCalendar‘GetHolidaysWithinDatesAsSunday	3	√
FCalendar‘GetNumberOfHolidaysWithinDates	10	√
FCalendar‘GetHolidaysWithinDatesNotSunday	21	√
Total Coverage		**100%**

2.5 FCharacter

文字（char）型に関わる関数を提供する。文字型で定義された機能以外の機能を定義する。

```
class
FCharacter
```

2.5.1 変換関数群

数字を整数に変換する

```
functions
public static
  AsDigit : char -> [int]
  AsDigit (c) ==
    cases c :
      '0' -> 0,
      '1' -> 1,
      '2' -> 2,
      '3' -> 3,
      '4' -> 4,
      '5' -> 5,
      '6' -> 6,
      '7' -> 7,
      '8' -> 8,
      '9' -> 9,
      others -> nil
    end;
```

英数字の辞書順序を返す。英数字以外の文字の場合は 256 を返す。

```
public static
  AsDictOrder : char -> int
  AsDictOrder (c) ==
    let DictOrderStr = "0123456789aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ",
        i = FString`Index (c) (DictOrderStr),
        undefinedSeq = 256 in
    cases true :
      (0 < i and i <= len DictOrderStr) -> i-1,
      others -> undefinedSeq
    end;
```

2.5.2 判定関数群

数字かどうか判定する。

```
public static
  IsDigit : char -> bool
  IsDigit (c) ==
    c in set elems "0123456789";
```

文字の辞書順序での大小を判定する。

```
public static
  LT : char +> char +> bool
  LT (c1)(c2) ==
    AsDictOrder (c1) < AsDictOrder (c2);
public static
  LE : char +> char +> bool
  LE (c1)(c2) ==
    LT (c1) (c2) or c1 = c2;
public static
  GT : char +> char +> bool
  GT (c1)(c2) ==
    LT (c2) (c1);
public static
  GE : char +> char +> bool
  GE (c1)(c2) ==
    not LT (c1) (c2)
end
```

FCharacter

Test Suite : vdm.tc

Class : FCharacter

Name	#Calls	Coverage
FCharacter‘GE	1	√
FCharacter‘GT	3	√
FCharacter‘LE	1	√
FCharacter‘LT	109	√
FCharacter‘AsDigit	11	√
FCharacter‘IsDigit	11	√
FCharacter‘AsDictOrder	226	√
Total Coverage		**100%**

2.6 FCharT

FCharacter のテストを行う。

```
class
FCharT
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 (),t3 ()] in
    FTestDriver`run (testcases);
```

2.6.1 数字を整数に変換

```
t1 : () +> FTestDriver`TestCase
t1() ==
  mk_FTestDriver`TestCase
      (
       "FCharT01 : \t 数字を整数に変換",
       let c = new FCharacter() in
                 '
       c.IsDigit('0') = true and
                 '
       c.IsDigit('1') = true and
                 '
       c.IsDigit('2') = true and
                 '
       c.IsDigit('3') = true and
                 '
       c.IsDigit('4') = true and
                 '
       c.IsDigit('5') = true and
                 '
       c.IsDigit('6') = true and
                 '
       c.IsDigit('7') = true and
                 '
       c.IsDigit('8') = true and
                 '
       c.IsDigit('9') = true and
                 '
       c.IsDigit('a') = false and
                 '
       c.AsDigit('0') = 0 and
                 '
       c.AsDigit('1') = 1 and
                 '
       c.AsDigit('2') = 2 and
                 '
       c.AsDigit('3') = 3 and
                 '
       c.AsDigit('4') = 4 and
                 '
       c.AsDigit('5') = 5 and
                 '
       c.AsDigit('6') = 6 and
                 '
       c.AsDigit('7') = 7 and
                 '
       c.AsDigit('8') = 8 and
                 '
       c.AsDigit('9') = 9 and
                 '
       c.AsDigit('a') = nil);
```

2.6.2 文字の辞書順序を返す

```
  t2 : () +> FTestDriver`TestCase
  t2() ==
    mk_FTestDriver`TestCase
        (
         "FCharT02 : \t 文字の辞書順序を返す",
         let c = new FCharacter() in
         c.AsDictOrder('0') = 0 and
         c.AsDictOrder('9') = 9 and
         c.AsDictOrder('a') = 10 and
         c.AsDictOrder('A') = 11 and
         c.AsDictOrder('z') = 60 and
         c.AsDictOrder('Z') = 61 and
         c.AsDictOrder('あ') = 256 and
         c.AsDictOrder('+') = 256);
```

2.6.3 文字の大小を比較する

```
  t3 : () +> FTestDriver`TestCase
  t3() ==
    mk_FTestDriver`TestCase
        (
         "FCharT03 : \t 文字の大小を比較する",
         let LT = FCharacter`LT,
             GT = FCharacter`GT,
             LE = FCharacter`LE,
             GE = FCharacter`GE in
         LT('a')('a') = false and
         GT('a')('a') = false and
         LT('1')('2') and
         GT('1')('0') and
         LT('9')('a') and
         GT('あ')('0') and
         LE('a')('0') = false and
         GE('a')('0') and
         FSequence`Fmap[char,bool](FCharacter`LT('5'))("456") = [false,false,true])
end
FCharT
```

2.7 FFunction

関数プログラミングに関わる関数を定義する。

```
class
FFunction
```

Funtil は、ある条件 p が真になるまで、初期値 x に関数 f を繰り返し適用する。

```
functions
public static
  Funtil[@T] : (@T +> bool) +> (@T +> @T) +> @T +> @T
  Funtil(p)(f)(x) ==
    if p(x)
    then x
    else Funtil[@T](p)(f)(f(x));
```

Fwhile は、ある条件 p が真である間、初期値 x に関数 f を繰り返し適用する。

```
public static
  Fwhile[@T] : (@T +> bool) +> (@T +> @T) +> @T +> @T
  Fwhile(p)(f)(x) ==
    if p(x)
    then Fwhile[@T](p)(f)(f(x))
    else x;
```

Seq は、引数 a に関数列 fs の関数を連続適用する。

```
public static
  Seq[@T] : seq of (@T +> @T) +> @T +> @T
  Seq(fs)(a) ==
    cases fs :
      [hf]^tf -> Seq[@T](tf)(hf(a)),
      [] -> a
    end
end
FFunction
```

Test Suite : vdm.tc

Class : FFunction

Name	#Calls	Coverage
FFunction‘Seq	8	√
FFunction‘Funtil	36	√
FFunction‘Fwhile	16	√
Total Coverage		**100%**

2.8 FunctionT

FFunction のテストを行う。

```
class
FFunctionT
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 ()] in
    FTestDriver`run (testcases);
```

2.8.1 Fwhile, Funtil を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FFunctionT01 : \t Fwhile, Funtil を検査する",
         let f1 = lambda x : int & x * 2,
             p1 = lambda x : int & x > 1000,
             p11 = lambda x : int & x <= 1000,
             f2 = lambda x : seq of char & x^"0",
             p2 = lambda x : seq of char & len x > 9,
             p21 = lambda x : seq of char & len x <= 9 in
         FFunction`Fwhile[int] (p11) (f1) (1) = 1024 and
         FFunction`Fwhile[seq of char] (p21) (f2) ("123456") = "1234560000" and
         FFunction`Funtil[int] (p1) (f1) (1) = 1024 and
         FFunction`Funtil[seq of char] (p2) (f2) ("123456") = "1234560000");
```

2.8.2 Seq を検査する

```
  t2 : () +> FTestDriver`TestCase
  t2() ==
    mk_FTestDriver`TestCase
        (
         "FFunctionT02 : \t Seq を検査する",
         let f1 = lambda x : int & x * 2,
             f2 = lambda x : int & x * 3,
             f3 = lambda x : int & x ** 2,
             関数列 1 = [f1, f2, f3],
             f10 = lambda x : seq of char & x^x,
             f11 = FSequence`Take[char](10),
             f12 = FSequence`Drop[char](4),
             関数列 2 = [f10, f11, f12] in
         FFunction`Seq[int](関数列 1)(2) = (2 * 2 * 3) ** 2 and
         FFunction`Seq[seq of char](関数列 2)("12345678") = "567812")
end
FFunctionT
```

2.9 FHashtable

ハッシュ表に関わる関数を定義する。

```
class
FHashtable
```

Put は、aKey と aValue の写をハッシュ表に追加する。

```
functions
public static
  Put[@T1, @T2] : (map @T1 to (map @T1 to @T2)) +> (@T1 +> @T1) +> @T1 +>
                  @T2 +>
                  (map @T1 to (map @T1 to @T2))
  Put (aHashtable)(aHashCode)(aKey)(aValue) ==
    let hashcode = aHashCode (aKey) in
    if hashcode in set dom aHashtable
    then aHashtable ++ {hashcode |-> (aHashtable (hashcode) ++ {aKey |-> aValue})}
    else aHashtable munion {hashcode |-> {aKey |-> aValue}};
```

PutAll は、写像の内容をハッシュ表に追加する。

```
public static
  PutAll[@T1, @T2] : (map @T1 to (map @T1 to @T2)) +> (@T1 +> @T1) +>
                     (map @T1 to @T2) +>
                     (map @T1 to (map @T1 to @T2))
  PutAll (aHashtable)(aHashCode)(aMap) ==
    PutAllAux[@T1, @T2] (aHashtable) (aHashCode) (aMap) (dom aMap);
public static
  PutAllAux[@T1, @T2] : (map @T1 to (map @T1 to @T2)) +> (@T1 +> @T1) +> (map @T1 to @T2) +>
                        set of @T1 +>
                        (map @T1 to (map @T1 to @T2))
  PutAllAux (aHashtable)(aHashCode)(aMap)(aKeySet) ==
    if aKeySet = {}
    then aHashtable
    else let aKey in set aKeySet in
         let 新 Hashtable = Put[@T1, @T2] (aHashtable) (aHashCode) (aKey) (aMap (aKey)) in
         PutAllAux[@T1, @T2] (新 Hashtable) (aHashCode) (aMap) (aKeySet \ {aKey});
```

Get は、aKey に対応する値を取り出す。

```
public static
```

```
  Get[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> (@T1 +> @T1) +> @T1 +> [@T2]
  Get(aHashtable)(aHashCode)(aKey) ==
    let hashcode = aHashCode(aKey) in
    if hashcode in set dom aHashtable
    then FMap`Get[@T1,@T2](aHashtable(hashcode))(aKey)
    else nil;
```

Remove は、key とそれに対応する値をハッシュ表から削除する。

```
public static
  Remove[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> (@T1 +> @T1) +> @T1 +> (map @T1 to (map @T1 to @T2))
  Remove(aHashtable)(aHashCode)(aKey) ==
    let hashcode = aHashCode(aKey) in
    {h |-> ({aKey} <-: aHashtable(hashcode))|h in set {hashcode}} munion
    {hashcode} <-: aHashtable;
```

Clear は、ハッシュ表をクリアーする。

```
public static
  Clear[@T1,@T2] : () +> (map @T1 to (map @T1 to @T2))
  Clear() ==
    ({ |-> });
```

KeySet は、ハッシュ表のすべての key の集合を返す。

```
public static
  KeySet[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> set of @T1
  KeySet(aHashtable) ==
    let aMapSet = rng aHashtable in
    if aMapSet <> {}
    then dunion {dom s|s in set aMapSet}
    else {};
```

ValueSet は、Hashtable のすべての値の集合を返す。

```
public static
  ValueSet[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> set of @T2
  ValueSet(aHashtable) ==
    let aMapSet = rng aHashtable in
    if aMapSet <> {}
    then dunion {rng s|s in set aMapSet}
    else {};
```

Size は、Hashtable 中の key の数を返す。

```
public static
  Size[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> nat
  Size(aHashtable) ==
    card KeySet[@T1,@T2](aHashtable);
```

IsEmpty は、Hashtable 中に key が無いか否かを返す。

```
public static
  IsEmpty[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> bool
  IsEmpty(aHashtable) ==
    KeySet[@T1,@T2](aHashtable) = {};
```

Contains は、与えられた aValue があるならば、true を返す。

```
public static
  Contains[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> @T2 +> bool
  Contains(aHashtable)(aValue) ==
    let aMapSet = rng aHashtable in
    if aMapSet <> {}
    then  exists aMap in set aMapSet & aValue in set rng aMap
    else false;
```

ContainsKey は、与えられた key があるならば、ｔｒｕｅを返す。

```
public static
  ContainsKey[@T1,@T2] : (map @T1 to (map @T1 to @T2)) +> @T1 +> bool
  ContainsKey(aHashtable)(aKey) ==
    let aMapSet = rng aHashtable in
    if aMapSet <> {}
    then  exists aMap in set aMapSet & aKey in set dom aMap
    else false
end
```

FHashtable

Test Suite : vdm.tc

Class : FHashtable

Name	#Calls	Coverage
FHashtable‘Get	9	√
FHashtable‘Put	21	√
FHashtable‘Size	4	√
FHashtable‘Clear	1	√
FHashtable‘KeySet	10	90%
FHashtable‘PutAll	4	√
FHashtable‘Remove	5	√
FHashtable‘IsEmpty	2	√
FHashtable‘Contains	7	91%
FHashtable‘ValueSet	4	90%
FHashtable‘PutAllAux	17	√
FHashtable‘ContainsKey	3	91%
Total Coverage		**97%**

2.10 FHashtableT

FHashtable のテストを行う。

```
class
FHashtableT
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 (),t3 (),t4 (),t5 (),t6 ()] in
    FTestDriver`run (testcases);
```

2.10.1 Contains, PutAll を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FHashtableT01 : \t Contains, PutAll を検査する",
         let aHashCode = lambda x : int & x mod 13,
             p1 = FHashtable`PutAll[int, seq of char] ({ |-> }) (aHashCode)
                     (
                      {1 |-> "Sahara", 2 |-> "Sato", 14 |-> "Sakoh"}),
             c1 = FHashtable`Contains[int, seq of char] (p1) in
         c1 ("Sahara") and
         c1 ("Sato") and
         c1 ("Sakoh") and
         c1 ("") = false);
```

2.10.2 Clear, Remove, ContainsKey を検査する

```
t2 : () +> FTestDriver`TestCase
t2() ==
  mk_FTestDriver`TestCase
      (
       "FHashtableT02 : \t Clear, Remove, ContainsKey を検査する",
       let aHashCode = lambda x : seq of char & if x = ""
                                                then ""
                                                else FSequence`Take[char](1)(x),
           h2 = FHashtable`PutAll[seq of char, int]({ |-> })(aHashCode)
                   (
                    {"a" |-> 1, "b" |-> 2, "c" |-> 3}),
           h3 = FHashtable`Clear[int, seq of char](),
           deletedh2 = FHashtable`Remove[seq of char, int](h2)(aHashCode)("b"),
           c1 = FHashtable`Contains[seq of char, int](deletedh2),
           ck1 = FHashtable`ContainsKey[seq of char, int](deletedh2) in
       h3 = { |-> } and
       FHashtable`Get[seq of char, int](deletedh2)(aHashCode)("b") = nil and
       c1(2) = false and
       c1(1) and
       c1(3) and
       ck1("b") = false and
       ck1("a") and
       ck1("c"));
```

2.10.3 Put, Get を検査する

```
t3 : () +> FTestDriver`TestCase
t3() ==
  mk_FTestDriver`TestCase
      (
       "FHashtableT03 : \t Put, Get を検査する",
       let aHashCode = lambda x : int & x mod 13,
           put = FHashtable`Put[int, seq of char],
           p1 = put({ |-> })(aHashCode)(1)("Sahara"),
           p2 = put(p1)(aHashCode)(2)("Bush"),
           p3 = put(p2)(aHashCode)(2)("Sato"),
           p4 = put(p3)(aHashCode)(14)("Sakoh"),
           get = FHashtable`Get[int, seq of char](p4),
           g = FHashtable`Get[int, seq of char](p4)(aHashCode) in
       get(aHashCode)(1) = "Sahara" and
       get(aHashCode)(2) = "Sato" and
       get(aHashCode)(14) = "Sakoh" and
       get(aHashCode)(99) = nil and
       FSequence`Fmap[int, seq of char](g)([1, 14]) = ["Sahara", "Sakoh"] and
       FSequence`Fmap[int, seq of char](g)([1, 2]) = ["Sahara", "Sato"]);
```

2.10.4 KeySet, ValueSet を検査する

```
t4 : () +> FTestDriver`TestCase
t4() ==
  mk_FTestDriver`TestCase
      (
       "FHashtableT04 : \t KeySet, ValueSet を検査する",
       let aHashCode = lambda x : int & x mod 13,
           put = FHashtable`Put[int, seq of char],
           p1 = put({ |-> })(aHashCode)(1)("Sahara"),
           p2 = put(p1)(aHashCode)(2)("Bush"),
           p3 = put(p2)(aHashCode)(2)("Sato"),
           p4 = put(p3)(aHashCode)(14)("Sakoh"),
           k = FHashtable`KeySet[int, seq of char],
           v = FHashtable`ValueSet[int, seq of char] in
       k(p1) = {1} and
       v(p1) = {"Sahara"} and
       k(p2) = {1, 2} and
       v(p2) = {"Sahara", "Bush"} and
       k(p4) = {1, 2, 14} and
       v(p4) = {"Sahara", "Sato", "Sakoh"});
```

2.10.5 hashCode が重複する場合を検査する

```
t5 : () +> FTestDriver`TestCase
t5() ==
  mk_FTestDriver`TestCase
      (
       "FHashtableT05 : \t hashCode が重複する場合を検査する",
       let aHashCode1 = lambda x : int & x mod 13,
           h1 = FHashtable`PutAll[int, seq of char]({ |-> })(aHashCode1)
                   (
                    {1 |-> "SaharaShin", 2 |-> "SatoKei", 14 |-> "SakohHiroshi", 27 |-> "NishikawaNoriko"})
           h2 = FHashtable`Remove[int, seq of char](h1)(aHashCode1)(14) in
       FHashtable`KeySet[int, seq of char](h2) = {1, 2, 27} and
       FHashtable`ValueSet[int, seq of char](h2) = {"SaharaShin", "SatoKei", "NishikawaNoriko"});
```

2.10.6 Size を検査する

```
  t6 : () +> FTestDriver`TestCase
  t6() ==
    mk_FTestDriver`TestCase
        (
         "FHashtableT06 : \t Size を検査する",
         let aHashCode1 = lambda x : int & x mod 13,
             remove = FHashtable`Remove[int, seq of char],
             h1 = FHashtable`PutAll[int, seq of char] ({ |-> }) (aHashCode1)
                     (
                      {1 |-> "SaharaShin", 2 |-> "SatoKei", 14 |-> "SakohHiroshi"}),
             h2 = remove (h1) (aHashCode1) (1),
             h3 = remove (h2) (aHashCode1) (2),
             h4 = remove (h3) (aHashCode1) (14),
             isempty = FHashtable`IsEmpty[int, seq of char],
             size = FHashtable`Size[int, seq of char] in
         isempty (h4) and
         size (h4) = 0 and
         isempty (h3) = false and
         size (h3) = 1 and
         size (h2) = 2 and
         size (h1) = 3)
end
FHashtableT
```

2.11 FInteger

整数型に関わる関数を定義する。

```
class
FInteger
```

2.11.1 変換関数群

AsString は、符号を考慮して、整数を数字文字列に変換する。

```
functions
public static
  AsString : int +> seq of char
  AsString (i) ==
    if i < 0
    then "-"^FInteger`AsStringAux (-i)
    else FInteger`AsStringAux (i);
```

AsStringAux は、符号を考慮せずに整数を数字文字列に変換する。

```
public static
  AsStringAux : nat +> seq of char
  AsStringAux (n) ==
    let m = n mod 10,
        d = n div 10 in
    cases d :
      0 -> AsChar (m),
      others -> AsStringAux (d)^FInteger`AsChar (m)
    end;
```

AsStringZ は、整数を ZZZ9.ZZ 形式の数字文字列に変換する（まだ小数点は考慮していない）。

```
public static
  AsStringZ : seq of char +> int +> seq of char
  AsStringZ (z)(i) ==
    let minus = '-',
        s = FString`SubStr (2) (len z) (z) in
    if i < 0
    then if z (1) = minus
         then [minus]^AsStringZAux (s) (-i) (true)
         else AsStringZAux (z) (-i) (true)
    else if z (1) = minus
         then AsStringZAux (s) (i) (true)
         else AsStringZAux (z) (i) (true);
```

AsStringZAux は、整数を ZZZ9.ZZ 形式の数字文字列に変換する補助関数（まだ小数点は考慮していな

い)。1桁で0の場合は、変換指定が"z"なら空白を返す。

```
public static
  AsStringZAux : seq of char +> nat +> bool +> seq of char
  AsStringZAux(z)(n)(toNowZero) ==
    let zLen = len z,
        z文字 = z(zLen),
        zStr = FString`SubStr(1)(zLen-1)(z),
        m = n mod 10,
        d = n div 10,
        wasZero = m = 0 and toNowZero and d <> 0 in
    cases zStr :
      [] -> FInteger`AsCharZ(z文字)(m)(wasZero),
      others -> FInteger`AsStringZAux(zStr)(d)(wasZero)^
                FInteger`AsCharZ(z文字)(m)(wasZero)
    end;
```

AsCharZは、ゼロサプレスを考慮して、整数を文字列の文字に変換する。変換指定文字の全部のケースは考慮していない。

```
public static
  AsCharZ : char +> nat +> bool +> seq of char|bool
  AsCharZ(zChar)(n)(toNowZero) ==
    cases n :
      0 ->
          if zChar in set {'z','Z'} and toNowZero
          then "0"
          elseif zChar = '0' or zChar = '9'
          then "0"
          else " ",
      1 -> "1",
      2 -> "2",
      3 -> "3",
      4 -> "4",
      5 -> "5",
      6 -> "6",
      7 -> "7",
      8 -> "8",
      9 -> "9",
      others -> false
    end;
```

AsCharは、1桁の整数iを文字列に変換する。iが1桁の整数でなければ、falseを返す。

```
public static
```

```
  AsChar : int +> seq of char|bool
  AsChar(i) ==
    cases i :
      0 -> "0",
      1 -> "1",
      2 -> "2",
      3 -> "3",
      4 -> "4",
      5 -> "5",
      6 -> "6",
      7 -> "7",
      8 -> "8",
      9 -> "9",
      others -> false
    end;
```

Gcd は、x, y の最大公約数を得る。

```
public static
  GCD : nat +> nat +> nat
  GCD(x)(y) ==
    if y = 0
    then x
    else GCD(y)(x rem y);
```

LCM は、x, y の最小公倍数を得る。

```
public static
  LCM : nat +> nat +> nat
  LCM(x)(y) ==
    cases mk_(x,y) :
      mk_(-,0) -> 0,
      mk_(0,-) -> 0,
      mk_(x,y) -> (x/GCD(x)(y)) * y
    end;
```

amod は商環（quotient ring）上での剰余計算を行う。例えば、暦の処理で１２で割った余りが０の場合１２を返したいときなどに使う。

```
public static
```

```
  amod : int +> int +> int
  amod(x)(y) ==
    let a = x mod y in
    if a = 0
    then y
    else x mod y
end
FInteger
```

Test Suite : vdm.tc

Class : FInteger

Name	#Calls	Coverage
FInteger‘GCD	21	√
FInteger‘LCM	5	√
FInteger‘amod	683	√
FInteger‘AsChar	37	√
FInteger‘AsCharZ	45	97%
FInteger‘AsString	6	√
FInteger‘AsStringZ	12	√
FInteger‘AsStringAux	26	√
FInteger‘AsStringZAux	45	√
Total Coverage		**99%**

2.12 FIntegerT

FInteger のテストを行う。

```
class
FIntegerT is subclass of FInteger
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 (),t3 ()] in
    FTestDriver`run (testcases);
```

2.12.1 AsString, AsStringZ, AsChar を検査する

```
t1 : () +> FTestDriver`TestCase
t1() ==
  mk_FTestDriver`TestCase
      (
       "FIntegerT`t1\t AsString, AsStringZ, AsChar を検査する",
       AsString(1234567890) = "1234567890" and
       AsString(-1234567890) = " - 1234567890" and
       AsStringZ("zzz9")(9900) = "9900" and
       AsStringZ("9")(0) = "0" and
       AsStringZ("z")(0) = "  " and
       AsStringZ("z")(9) = "9" and
       AsStringZ("zzz9")(9) = "   9" and
       AsStringZ("0009")(9) = "0009" and
       AsStringZ(" - 0009")(9) = "0009" and
       AsStringZ(" - zzz9")(-9999) = " - 9999" and
       AsStringZ(" - zzz9")(-9) = " -    9" and
       AsStringZ(" - zzzzzzzzz9")(-1234567890) = " - 1234567890" and
       AsStringZ("zzz9")(-9999) = "9999" and
       AsStringZ("zzz9")(-9) = "   9" and
       AsString(0) = "0" and
       AsChar(0) = "0" and
       AsChar(1) = "1" and
       AsChar(2) = "2" and
       AsChar(3) = "3" and
       AsChar(4) = "4" and
       AsChar(5) = "5" and
       AsChar(6) = "6" and
       AsChar(7) = "7" and
       AsChar(8) = "8" and
       AsChar(9) = "9" and
       AsChar(10) = false);
```

2.12.2 Gcd, Lcm を検査する

```
  t2 : () +> FTestDriver`TestCase
  t2() ==
    mk_FTestDriver`TestCase
        (
         "FIntegerT`t2 : \t Gcd, Lcm を検査する",
         let gcd = GCD(24),
             lcm = LCM(7) in
         FSequence`Fmap[nat, nat](gcd)([36, 48, 16]) = [12, 24, 8] and
         FSequence`Fmap[nat, nat](lcm)([3, 4, 5]) = [21, 28, 35] and
         LCM(7)(0) = 0 and
         LCM(0)(3) = 0);
```

2.12.3 amod を検査する

```
  t3 : () +> FTestDriver`TestCase
  t3() ==
    mk_FTestDriver`TestCase
        (
         "FIntegerT`t3 : \t amod を検査する",
         amod(0)(12) = 12 and
         amod(12)(12) = 12 and
         amod(24)(12) = 12 and
         amod(1)(12) = 1 and
         amod(2)(12) = 2 and
         amod(11)(12) = 11 and
         amod(13)(12) = 1 and
         amod(23)(12) = 11)
end
FIntegerT
```

2.13 FJapaneseCalendar

日本の暦に関わる関数を定義する。

```
class
FJapaneseCalendar is subclass of FCalendar
values
public
  DiffBetweenGMTandJST = 9;
public
  DiffBetweenSeirekiAndHoureki = 1988
```

GetHolidays は、指定した年 yyyy（2000 年以降）の日本の休日の集合を返す。

```
functions
public static
```

```
  GetHolidays : int +> set of Date
  GetHolidays (yyyy) ==
    let Seijin = GetNthDayOfWeekOfMonth ( <Mon> ) (2) (1) (yyyy),
        Umi =
            if yyyy >= 2003
            then GetNthDayOfWeekOfMonth ( <Mon> ) (3) (7) (yyyy)
            else DateFromInt (yyyy) (7) (20),
        Keirou =
            if yyyy >= 2003
            then GetNthDayOfWeekOfMonth ( <Mon> ) (3) (9) (yyyy)
            else DateFromInt (yyyy) (9) (15),
        Taiiku = GetNthDayOfWeekOfMonth ( <Mon> ) (2) (10) (yyyy),
        NationalHoliday = {
                          DateFromInt (yyyy) (1) (1),
                          Seijin,
                          DateFromInt (yyyy) (2) (11),
                          GetVernalEquinoxInJST (yyyy),
                          DateFromInt (yyyy) (4) (29),
                          DateFromInt (yyyy) (5) (3),
                          DateFromInt (yyyy) (5) (4),
                          DateFromInt (yyyy) (5) (5),
                          Umi,
                          Keirou,
                          GetAutumnalEquinoxInJST (yyyy),
                          Taiiku,
                          DateFromInt (yyyy) (11) (3),
                          DateFromInt (yyyy) (11) (23),
                          DateFromInt (yyyy) (12) (23)},
        holidayInLieu = {d + 1|d in set NationalHoliday & IsSunday (d)} in
      NationalHoliday union holidayInLieu;
```

GetDateInJST は、日本標準時基準の日付を得る。

```
public static
  GetDateInJST : Date +> Date
  GetDateInJST (d) ==
    GetDateInST (DiffBetweenGMTandJST) (d);
private static
```

```
  AsStringAux : int +> seq of char
  AsStringAux (i) ==
    let str = FInteger`AsString in
    if i >= 10
    then str (i)
    else " "^str (i);
```

GetJapaneseYearAsString は、平成以後の和暦日付文字列を得る

```
public static
  GetJapaneseYearAsString : Date +> seq of char
  GetJapaneseYearAsString (d) ==
    let asString = FInteger`AsString,
        JapaneseYear = Year (d)-DiffBetweenSeirekiAndHoureki,
        m = Month (d),
        aDate = Day (d),
        YY = asString (JapaneseYear),
        MM = AsStringAux (m),
        DD = AsStringAux (aDate) in
    YY^MM^DD;
```

GetVernalEquinoxInJST は、yyyy 年の日本標準時の春分を得る。

```
public static
  GetVernalEquinoxInJST : int +> Date
  GetVernalEquinoxInJST (yyyy) ==
    GetDateInJST (GetVernalEquinoxInGMT (yyyy));
```

GetSummerSolsticeInJST は、yyyy 年の日本標準時の夏至を得る。

```
public static
  GetSummerSolsticeInJST : int +> Date
  GetSummerSolsticeInJST (yyyy) ==
    GetDateInJST (GetSummerSolsticeInGMT (yyyy));
```

GetAutumnalEquinoxInJST は、yyyy 年の日本標準時の秋分を得る。

```
public static
  GetAutumnalEquinoxInJST : int +> Date
  GetAutumnalEquinoxInJST (yyyy) ==
    GetDateInJST (GetAutumnalEquinoxInGMT (yyyy));
```

GetWinterSolsticeInJST は、yyyy 年の日本標準時の冬至を得る。

```
public static
  GetWinterSolsticeInJST : int +> Date
  GetWinterSolsticeInJST (yyyy) ==
    GetDateInJST (GetWinterSolsticeInGMT (yyyy))
end
```

FJapaneseCalendar

Test Suite : vdm.tc

Class : FJapaneseCalendar

Name	#Calls	Coverage
FJapaneseCalendar‘AsStringAux	2	√
FJapaneseCalendar‘GetHolidays	143	√
FJapaneseCalendar‘GetDateInJST	298	√
FJapaneseCalendar‘GetVernalEquinoxInJST	146	√
FJapaneseCalendar‘GetSummerSolsticeInJST	3	√
FJapaneseCalendar‘GetWinterSolsticeInJST	3	√
FJapaneseCalendar‘GetAutumnalEquinoxInJST	146	√
FJapaneseCalendar‘GetJapaneseYearAsString	1	√
Total Coverage		**100%**

2.14 FJapaneseCalendarT

FJapaneseCalendar のテストを行う。

```
class
FJapaneseCalendarT is subclass of FJapaneseCalendar
functions
public static
  run : () -> bool
  run () ==
    let testcases =
            [t1 (),t2 (),t3 (),t4 (),t5 (),t6 (),t7 (),t8 (),t9 (),t10 (),
             t11 ()] in
    FTestDriver`run (testcases);
```

2.14.1 「GetHolidaysWithinDays」を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FCalendarT`t1 : \t「GetHolidaysWithinDays」を検査する",
         let d = FCalendar`DateFromInt,
             g = GetHolidaysWithinDates (FJapaneseCalendar`GetHolidays) in
         g (d (2004) (4) (28)) (d (2004) (4) (29)) = {d (2004) (4) (29)} and
         g (d (2004) (4) (28)) (d (2004) (5) (2)) = {d (2004) (4) (29)} and
         g (d (2004) (4) (28)) (d (2004) (5) (3)) = {d (2004) (4) (29),d (2004) (5) (3)} and
         g (d (2004) (4) (28)) (d (2004) (5) (4)) = {d (2004) (4) (29),d (2004) (5) (3),d (2004) (5) (4)} and
         g (d (2004) (4) (28)) (d (2004) (5) (5)) = {d (2004) (4) (29),d (2004) (5) (3),d (2004) (5) (4),d (2004) (5
         g (d (2004) (4) (29)) (d (2004) (5) (5)) = {d (2004) (4) (29),d (2004) (5) (3),d (2004) (5) (4),d (2004) (5
         g (d (2004) (4) (30)) (d (2004) (5) (5)) = {d (2004) (5) (3),d (2004) (5) (4),d (2004) (5) (5)} and
         g (d (2004) (1) (1)) (d (2004) (12) (31)) =
         {d (2004) (1) (1),d (2004) (1) (12),d (2004) (2) (11),d (2004) (3) (20),d (2004) (4) (29),
          d (2004) (5) (3),d (2004) (5) (4),d (2004) (5) (5),d (2004) (7) (19),d (2004) (9) (20),d (2004) (9) (23)
          d (2004) (10) (11),d (2004) (11) (3),d (2004) (11) (23),d (2004) (12) (23)} and
         g (d (2005) (3) (1)) (d (2005) (3) (31)) = {d (2005) (3) (20),d (2005) (3) (21)});
```

2.14.2 「GetHolidaysWithinDatesNotSunday」を検査する

```
t2 : () +> FTestDriver`TestCase
t2() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t2 : \t「GetHolidaysWithinDatesNotSunday」を検査する",
       let d = FCalendar`DateFromInt,
           g = GetHolidaysWithinDatesNotSunday(FJapaneseCalendar`GetHolidays) in
       g(d(2004)(1)(1))(d(2004)(12)(31)) =
       {d(2004)(1)(1),d(2004)(1)(12),d(2004)(2)(11),d(2004)(3)(20),d(2004)(4)(29),
        d(2004)(5)(3),d(2004)(5)(4),d(2004)(5)(5),d(2004)(7)(19),d(2004)(9)(20),d(2004)(9)(23)
        d(2004)(10)(11),d(2004)(11)(3),d(2004)(11)(23),d(2004)(12)(23)} and
       g(d(2005)(3)(1))(d(2005)(3)(31)) = {d(2005)(3)(21)} and
       g(d(2006)(1)(1))(d(2006)(1)(31)) = {d(2006)(1)(2),d(2006)(1)(9)});
```

2.14.3 「GetHolidaysWithinDatesAsSunday」を検査する

```
t3 : () +> FTestDriver`TestCase
t3() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t3 : \t「GetHolidaysWithinDatesAsSunday」を検査する",
       let d = FCalendar`DateFromInt,
           g = GetHolidaysWithinDatesAsSunday(FJapaneseCalendar`GetHolidays) in
       g(d(2004)(1)(1))(d(2004)(12)(31)) = {} and
       g(d(2005)(3)(1))(d(2005)(3)(31)) = {d(2005)(3)(20)} and
       g(d(2006)(1)(1))(d(2006)(1)(31)) = {d(2006)(1)(1)});
```

2.14.4 「GetNumberOfHolidaysWithinDates」を検査する

```
t4 : () +> FTestDriver`TestCase
t4() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t4 : \t「GetNumberOfHolidaysWithinDates」を検査する",
       let d = FCalendar`DateFromInt,
           g = GetNumberOfHolidaysWithinDates(FJapaneseCalendar`GetHolidays) in
       g(d(2004)(4)(28))(d(2004)(4)(29)) = 1 and
       g(d(2004)(4)(28))(d(2004)(5)(2)) = 1 and
       g(d(2004)(4)(28))(d(2004)(5)(3)) = 2 and
       g(d(2004)(4)(28))(d(2004)(5)(4)) = 3 and
       g(d(2004)(4)(28))(d(2004)(5)(5)) = 4 and
       g(d(2004)(4)(29))(d(2004)(5)(5)) = 4 and
       g(d(2004)(4)(30))(d(2004)(5)(5)) = 3 and
       g(d(2004)(1)(1))(d(2004)(12)(31)) = 15 and
       g(d(2005)(3)(1))(d(2005)(3)(31)) = 2 and
       g(d(2006)(1)(1))(d(2006)(1)(31)) = 3);
```

2.14.5 「GetNumberOfDayOff」を検査する

```
t5 : () +> FTestDriver`TestCase
t5() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t5 : \t「GetNumberOfDayOff」を検査する",
       let d = FCalendar`DateFromInt,
           g = GetNumberOfDayOff(FJapaneseCalendar`GetHolidays) in
       g(d(2004)(4)(28))(d(2004)(4)(29)) = 1 and
       g(d(2004)(4)(28))(d(2004)(5)(2)) = 2 and
       g(d(2004)(4)(28))(d(2004)(5)(3)) = 3 and
       g(d(2004)(4)(28))(d(2004)(5)(4)) = 4 and
       g(d(2004)(4)(28))(d(2004)(5)(5)) = 5 and
       g(d(2004)(4)(29))(d(2004)(5)(5)) = 5 and
       g(d(2004)(4)(30))(d(2004)(5)(5)) = 4 and
       g(d(2005)(3)(1))(d(2005)(3)(31)) = 5 and
       g(d(2006)(1)(1))(d(2006)(1)(31)) = 7);
```

2.14.6 「GetNumberOfDayOff1」を検査する

```
t6 : () +> FTestDriver`TestCase
t6() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t6 : \t「GetNumberOfDayOff1」を検査する",
       let d = FCalendar`DateFromInt,
           g = GetNumberOfDayOff1(FJapaneseCalendar`GetHolidays) in
       g(d(2004)(4)(28))(d(2004)(4)(29)) = 1 and
       g(d(2004)(4)(28))(d(2004)(5)(2)) = 2 and
       g(d(2004)(4)(28))(d(2004)(5)(3)) = 3 and
       g(d(2004)(4)(28))(d(2004)(5)(4)) = 4 and
       g(d(2004)(4)(28))(d(2004)(5)(5)) = 5 and
       g(d(2004)(4)(29))(d(2004)(5)(5)) = 4 and
       g(d(2004)(4)(30))(d(2004)(5)(5)) = 4 and
       g(d(2005)(3)(1))(d(2005)(3)(31)) = 5 and
       g(d(2006)(1)(1))(d(2006)(1)(31)) = 6);
```

2.14.7 休日を考慮した加減算 (+-1) を検査する

```
t7 : () +> FTestDriver`TestCase
t7() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t7 : \t 休日を考慮した加減算 (+ − 1) を検査する",
       let d = FCalendar`DateFromInt,
           n = BusinessDateToFuture(FJapaneseCalendar`GetHolidays),
           p = BusinessDateToPast(FJapaneseCalendar`GetHolidays) in
       n(d(2004)(4)(29)) = d(2004)(4)(30) and
       p(d(2004)(4)(29)) = d(2004)(4)(28) and
       n(d(2004)(5)(1)) = d(2004)(5)(6) and
       n(d(2004)(5)(2)) = d(2004)(5)(6) and
       p(d(2004)(5)(5)) = d(2004)(4)(30));
```

2.14.8 休日を考慮した加減算を検査する

```
t8 : () +> FTestDriver`TestCase
t8() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t8 : \t 休日を考慮した加減算を検査する",
       let d = FCalendar`DateFromInt,
           n = BusinessDateToFuture(FJapaneseCalendar`GetHolidays),
           p = BusinessDateToPast(FJapaneseCalendar`GetHolidays),
           a = AddBusinessDays(FJapaneseCalendar`GetHolidays),
           s = SubtractBusinessDays(FJapaneseCalendar`GetHolidays) in
       n(d(2004)(4)(28))=a(d(2004)(4)(28))(0) and
       p(d(2004)(4)(30))=s(d(2004)(4)(30))(0) and
       n(d(2004)(4)(29))=d(2004)(4)(30) and
       n(d(2004)(5)(1))=d(2004)(5)(6) and
       p(d(2004)(5)(6))=d(2004)(5)(6) and
       p(d(2004)(5)(5))=d(2004)(4)(30) and
       s(d(2004)(5)(6))(1)=d(2004)(4)(30) and
       a(d(2004)(5)(1))(1)=d(2004)(5)(7) and
       s(d(2004)(5)(1))(-1)=d(2004)(4)(30) and
       a(d(2004)(5)(6))(-1)=d(2004)(5)(6) and
       s(d(2004)(5)(6))(1)=d(2004)(4)(30) and
       s(d(2004)(5)(6))(6)=d(2004)(4)(22) and
       a(d(2004)(4)(22))(6)=d(2004)(5)(6));
```

2.14.9 休日の判定を検査する

```
t9 : () +> FTestDriver`TestCase
t9() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t9 : \t 休日の判定を検査する",
       let d = FCalendar`DateFromInt,
           h = IsHoliday(FJapaneseCalendar`GetHolidays),
           f = IsDayOff(FJapaneseCalendar`GetHolidays) in
       h(d(2004)(4)(29)) and
       h(d(2004)(5)(2)) = false and
       h(d(2004)(5)(3)) and
       h(d(2004)(5)(4)) and
       h(d(2004)(5)(5)) and
       h(d(2004)(5)(6)) = false and
       f(d(2004)(5)(2)) and
       f(d(2004)(5)(9)) and
       f(d(2005)(3)(19)) = false and
       f(d(2005)(3)(20)) and
       f(d(2005)(3)(21)) and
       f(d(2005)(3)(22)) = false);
```

2.14.10 「和暦日付文字列を得る」を検査する

```
t10 : () +> FTestDriver`TestCase
t10() ==
  mk_FTestDriver`TestCase
      (
       "FCalendarT`t10 : \t「和暦日付文字列を得る」を検査する",
       GetJapaneseYearAsString(DateFromInt(2004)(2)(29)) = "16 229");
```

2.14.11 春分・夏至・秋分・冬至を検査する

```
  t11 : () +> FTestDriver`TestCase
  t11() ==
    mk_FTestDriver`TestCase
        (
         "FCalendarT`t11 : \t 春分・夏至・秋分・冬至を検査する",
         let d = FCalendar`DateFromInt in
         GetVernalEquinoxInJST(2004) = d(2004)(3)(20) and
         GetVernalEquinoxInJST(2003) = d(2003)(3)(21) and
         GetVernalEquinoxInJST(2020) = d(2020)(3)(20) and
         GetSummerSolsticeInJST(2004) = d(2004)(6)(21) and
         GetSummerSolsticeInJST(2003) = d(2003)(6)(22) and
         GetSummerSolsticeInJST(2020) = d(2020)(6)(21) and
         GetAutumnalEquinoxInJST(2004) = d(2004)(9)(23) and
         GetAutumnalEquinoxInJST(2003) = d(2003)(9)(23) and
         GetAutumnalEquinoxInJST(2020) = d(2020)(9)(22) and
         GetWinterSolsticeInJST(2004) = d(2004)(12)(22) and
         GetWinterSolsticeInJST(2003) = d(2003)(12)(22) and
         GetWinterSolsticeInJST(2020) = d(2020)(12)(21))
end
FJapaneseCalendarT
```

2.15 FHashtable

写像に関わる関数を定義する。

```
class
FMap
```

Get は、写像に aKey があれば、対応する値域の値を返し、無ければ nil を返す。

```
functions
public static
  Get[@T1,@T2] : map @T1 to @T2 +> @T1 +> [@T2]
  Get(aMap)(aKey) ==
    if aKey in set dom aMap
    then aMap(aKey)
    else nil;
```

Contains は、aValue が写像 aMap の値域に含まれているか否かを返す。

```
public static
  Contains[@T1,@T2] : map @T1 to @T2 +> @T2 +> bool
  Contains(aMap)(aValue) ==
    aValue in set rng aMap;
```

ContainsKey は、aKey が写像 aMap の定義域に含まれているか否かを返す。

```
public static
  ContainsKey[@T1,@T2] : map @T1 to @T2 +> @T1 +> bool
  ContainsKey(aMap)(aKey) ==
    aKey in set dom aMap
end
FMap
```

Test Suite : vdm.tc

Class : FMap

Name	#Calls	Coverage
FMap‘Get	12	√
FMap‘Contains	8	√
FMap‘ContainsKey	8	√
Total Coverage		**100%**

2.16 FMapT

FMap のテストを行う。

```
class
FMapT
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 ()] in
    FTestDriver`run (testcases);
```

2.16.1 Get を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FMapT01 : \t Get を検査する",
         let m1 = {1 |-> "KeiSato",19 |-> "ShinSahara",20 |-> "HisoshiSako"},
             m2 = {"KeiSato" |-> 1,"ShinSahara" |-> 19,"HisoshiSako" |-> 20},
             get1 = FMap`Get[int,seq of char],
             get2 = FMap`Get[seq of char,int] in
         get1 (m1) (19) = "ShinSahara" and
         get1 (m1) (2) = nil and
         get2 (m2) ("ShinSahara") = 19 and
         get2 (m2) ("KoizumiBoo") = nil);
```

2.16.2 Contains, ContainsKey を検査する

```
  t2 : () +> FTestDriver`TestCase
  t2() ==
    mk_FTestDriver`TestCase
        (
         "FMapT01 : \t Contains, ContainsKey を検査する",
         let m1 = {1 |-> "KeiSato", 19 |-> "ShinSahara", 20 |-> "HisoshiSako"},
             m2 = {"KeiSato" |-> 1, "ShinSahara" |-> 19, "HisoshiSako" |-> 20},
             c1 = FMap`Contains[int, seq of char],
             k1 = FMap`ContainsKey[int, seq of char],
             c2 = FMap`Contains[seq of char, int],
             k2 = FMap`ContainsKey[seq of char, int] in
         c1(m1)("KeiSato") and c1(m1)("ShinSahara") and c1(m1)("HisoshiSako") and
         c1(m1)("KoizumiBoo") = false and
         k1(m1)(1) and
         k1(m1)(19) and k1(m1)(20) and
         not k1(m1)(99) and
         c2(m2)(1) and
         c2(m2)(19) and c2(m2)(20) and
         c2(m2)(30) = false and
         k2(m2)("KeiSato") and
         k2(m2)("ShinSahara") and k2(m2)("HisoshiSako") and
         k2(m2)("KoizumiBoo") = false)
end
FMapT
```

2.17 FNumber

整数、実数などに共通の、数に関わる関数を定義する。

```
class
FNumber
```

2.17.1 計算に関する関数群

IsComputable は、計算可能か否かを返す。

```
functions
public static
  IsComputable[@T] : @T +> bool
  IsComputable (n) ==
    is_(n, int) or is_(n, nat) or is_(n, nat1) or is_(n, real) or is_(n, rat);
```

Min は、関数 f で指定された順序で、n1, n2 の最小値を返す。

```
public static
  Min[@T] : (@T +> @T +> bool) +> @T +> @T +> @T
  Min (f)(n1)(n2) ==
    if f (n1) (n2)
    then n2
    else n1
  pre  IsComputable[@T] (n1) and IsComputable[@T] (n2) ;
```

Max は、関数 f で指定された順序で、n1, n2 の最大値を返す。

```
public static
  Max[@T] : (@T +> @T +> bool) +> @T +> @T +> @T
  Max (f)(n1)(n2) ==
    if f (n1) (n2)
    then n1
    else n2
  pre  IsComputable[@T] (n1) and IsComputable[@T] (n2) ;
```

2.17.2 数の大小を判定する関数群

関数型の引数として使用する。

```
public static
  LT : rat +> rat +> bool
  LT (c1)(c2) ==
    c1 < c2;
public static
```

```
  LE : rat +> rat +> bool
  LE(c1)(c2) ==
    c1 <= c2;
public static
  GT : rat +> rat +> bool
  GT(c1)(c2) ==
    c1 > c2;
public static
  GE : rat +> rat +> bool
  GE(c1)(c2) ==
    c1 >= c2
end
```

FNumber

Test Suite : vdm.tc

Class : FNumber

Name	#Calls	Coverage
FNumber‘GE	6	√
FNumber‘GT	211	√
FNumber‘LE	6	√
FNumber‘LT	6	√
FNumber‘Max	102	√
FNumber‘Min	103	√
FNumber‘IsComputable	416	√
Total Coverage		**100%**

2.18 FNumberT

FNumber のテストを行う。

```
class
FNumberT is subclass of FNumber
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 (),t3 ()] in
    FTestDriver`run (testcases);
```

2.18.1 Min, Max を検査する

```
public static
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FNumberT01 : \t Min, Max を検査する",
         Min[int] (FNumber`GT) (-3) (4) = -3 and
         Min[int] (FNumber`GT) (4) (-3) = -3 and
         Min[nat] (FNumber`GT) (2) (10) = 2 and
         Min[int] (FNumber`GT) (0) (0) = 0 and
         Max[real] (FNumber`GT) (0.001) (-0.001) = 0.001 and
         Max[real] (FNumber`GT) (-0.001) (0.001) = 0.001 and
         Max[real] (FNumber`GT) (0) (0) = 0);
```

2.18.2 IsComputable を検査する

```
t2 : () +> FTestDriver`TestCase
t2() ==
  mk_FTestDriver`TestCase
      (
       "FNumberT02 : \t IsComputable を検査する",
       IsComputable[char]('a') = false and
       IsComputable[int](-9) = true and
       IsComputable[nat](0) = true and
       IsComputable[nat1](1) = true and
       IsComputable[real](1.234) = true and
       IsComputable[rat](1.234) = true);
```

2.18.3 GT, GE, LT, LE を検査する

```
  t3 : () +> FTestDriver`TestCase
  t3() ==
    mk_FTestDriver`TestCase
        (
         "FNumberT03 : \t GT, GE, LT, LE を検査する",
         LT(1)(2) and
         LT(2)(2) = false and
         LT(3)(2) = false and
         LT(1.1)(2.1) and
         LT(2.1)(2.1) = false and
         LT(3.1)(3) = false and
         LE(1)(2) and
         LE(2)(2) and
         LE(3)(2) = false and
         LE(1.9)(2) and
         LE(2.2)(2.2) and
         LE(3)(2.999) = false and
         GT(2)(1) and
         GT(2)(2) = false and
         GT(1)(2) = false and
         GT(2)(1.999) and
         GT(2.999)(2.999) = false and
         GT(2)(2) = false and
         GE(2)(1) and
         GE(2)(2) and
         GE(2)(3) = false and
         GE(2)(1.9) and
         GE(2.999999)(2.999999) and
         GE(3)(3) = false)
end
FNumberT
```

2.19 FQueueT

FQueue のテストを行う。

```
class
FQueueT
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 ()] in
    FTestDriver`run (testcases);
```

2.19.1 Empty, EnQueue, DeQueue, Top, FromList, ToList, IsEmpty を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1() ==
    mk_FTestDriver`TestCase
        (
         "FQueueT01 : \t Empty, EnQueue, DeQueue, Top, FromList, ToList, IsEmpty を検査する",
         let q0 = FQueue`Empty[int](),
             q1 = FQueue`EnQueue[int](1)(q0),
             q2 = FQueue`EnQueue[int](2)(q1),
             q3 = FQueue`EnQueue[int](3)(q2),
             h1 = FQueue`Top[int](q3),
             q4 = FQueue`DeQueue[int](q3),
             q5 = FQueue`EnQueue[int](4)(q4),
             q6 = FQueue`EnQueue[int](5)(q5),
             q7 = FQueue`DeQueue[int](q6),
             q8 = FQueue`DeQueue[int](q7),
             q9 = FQueue`DeQueue[int](q8),
             q10 = FQueue`DeQueue[int](q9),
             h2 = FQueue`Top[int](q10),
             q11 = FQueue`DeQueue[int](q10),
             q12 = FQueue`FromList[char]("Sahara Shin")(FQueue`Empty[char]()) in
         FQueue`IsEmpty[int](q0) and q0 = mk_([],[]) and
         FQueue`ToList[int](q1) = [1] and
         q1 = mk_([],[1]) and
         FQueue`ToList[int](q2) = [1,2] and
         q2 = mk_([],[2,1]) and
         FQueue`ToList[int](q3) = [1,2,3] and
         q3 = mk_([],[3,2,1]) and
         h1 = 1 and
         FQueue`ToList[int](q4) = [2,3] and
         q4 = mk_([2,3],[]) and
         FQueue`ToList[int](q5) = [2,3,4] and
         q5 = mk_([2,3],[4]) and
         FQueue`ToList[int](q6) = [2,3,4,5] and
         q6 = mk_([2,3],[5,4]) and
         FQueue`ToList[int](q7) = [3,4,5] and
         q7 = mk_([3],[5,4]) and
         FQueue`ToList[int](q8) = [4,5] and
         q8 = mk_([],[5,4]) and
         FQueue`ToList[int](q9) = [5] and
         q9 = mk_([5],[]) and
         FQueue`ToList[int](q10) = [] and
         FQueue`IsEmpty[int](q10) and q10 = mk_([],[]) and
         h2 = nil and
         q11 = nil and
         FQueue`ToList[char](q12) = "Sahara Shin" and
```

```
end
FQueueT
```

2.20 FProduct

組に関わる関数を定義する。

```
class
FProduct
```

Curry は、カリー化を行う。

```
functions
public static
  Curry[@T1, @T2, @T3] : (@T1 * @T2 +> @T3) +> @T1 +> @T2 +> @T3
  Curry (f)(x)(y) ==
    f (x, y);
```

Uncurry は、カリー化の逆を行う。

```
public static
  Uncurry[@T1, @T2, @T3] : (@T1 +> @T2 +> @T3) +> @T1 * @T2 +> @T3
  Uncurry (f)(x, y) ==
    f (x) (y)
end
FProduct
```

Test Suite : vdm.tc

Class : FProduct

Name	#Calls	Coverage
FProduct‘Curry	3	√
FProduct‘Uncurry	3	√
Total Coverage		**100%**

2.21 FProductT

FProduct のテストを行う。

```
class
FProductT
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 ()] in
    FTestDriver`run (testcases);
```

2.21.1 Curry, Uncurry を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FProductT01 : \t Curry, Uncurry を検査する",
         let lt = lambda x : seq of char, y : seq of char & len x < len y,
             lt2 = lambda x : int, y : int & x < y in
         FProduct`Curry[seq of char, seq of char, bool] (lt) ("abc") ("abcd") and
         FProduct`Curry[seq of char, seq of char, bool] (lt) ("abcde") ("abcd") = false and
         FProduct`Curry[int, int, bool] (lt2) (3) (4) and
         FProduct`Uncurry[seq of char, seq of char, bool] (FString`LT) ("abc", "abcd") and
         FProduct`Uncurry[seq of char, seq of char, bool] (FString`LT) ("abcde", "abcd")  =
false and
         FProduct`Uncurry[seq of char, seq of char, bool] (FString`LE) ("3", "4"))
end
FProductT
```

2.22 FReal

実数（real）に関わる関数を提供する。実数や数値演算であらかじめ定義された機能以外の機能を定義する。

```
class
FReal
values
  ErrRng = 0;
  h = 0.00001
```

Figures は、実数の（小数点や小数点以下の桁数を含む）桁数を返す。

```
functions
public static
  Figures : real +> nat
  Figures (x) ==
    let i = floor (x) in
    if x = i
    then FiguresOfInteger (i)
    else FiguresOfInteger (i) + 1 + NumberOfDecimalPlaces (x);
```

FiguresOfInteger は、整数 i の整数部桁数を返す。

```
public static
  FiguresOfInteger : int +> nat
  FiguresOfInteger (i) ==
    FiguresOfIntegerAux (1) (i)
  post if i = 0
       then RESULT = 1
       else 10 ** (RESULT-1) <= abs (i) and abs (i) < 10 ** RESULT ;
```

FiguresOfIntegerAux は、整数 i の整数部桁数 n を返す補助関数。

```
  FiguresOfIntegerAux : nat +> int +> nat
  FiguresOfIntegerAux (n)(i) ==
    let q = i div 10 in
    cases q :
      0 -> n,
      others -> FReal`FiguresOfIntegerAux (n + 1) (q)
    end;
```

IsDecimalPlacesN は、実数 x が小数点以下 n 桁か否かを示す。

```
public static
```

```
  IsDecimalPlacesN : nat +> real +> bool
  IsDecimalPlacesN(n)(x) ==
    let f = lambda y : real & y * 10 ** n in
    f(x) = floor f(x);
```

NumberOfDecimalPlaces は、実数 x が小数点以下何桁かを返す。

```
public static
  NumberOfDecimalPlaces : real +> nat
  NumberOfDecimalPlaces(x) ==
    NumberOfDecimalPlacesAux(0)(x)
  post IsDecimalPlacesN(RESULT)(x) ;
```

NumberOfDecimalPlacesAux は、実数 x が小数点以下何桁かを返す補助関数。

```
  NumberOfDecimalPlacesAux : nat +> real +> nat
  NumberOfDecimalPlacesAux(n)(x) ==
    if x = floor (x)
    then n
    else NumberOfDecimalPlacesAux(n + 1)(x * 10);
```

Round は、実数 r を小数点以下 n 桁で四捨五入する。

```
public static
  Round : real +> nat +> real
  Round(r)(n) ==
    let m = 10 ** n in
    floor (r * m + 0.5)/m
  pre  r >= 0 ;
```

Differentiate は、関数 f(x) を x で微分する。

```
public static
  Differentiate : (real +> real) +> real +> real
  Differentiate(f)(x) ==
    (f(x + h)-f(x))/h;
```

NewtonOfEquation は、ニュートン法で方程式を解く。

```
public static
  NewtonOfEquation : (real +> real) +> real +> real
  NewtonOfEquation(f)(x) ==
    let endCndition = lambda y : real & abs (f(y)) < ErrRng,
        next = lambda y : real & y-(f(y)/Differentiate(f)(y)) in
    FFunction`Funtil[real](endCndition)(next)(x);
```

Root は、ニュートン法で実数 x の平方根を求める。ニュートン法のテストのために作成した。

```
public static
```

```
  Root : real +> real
  Root (x) ==
    let f = lambda y : real & y ** 2-x in
    NewtonOfEquation (f) (x)
  pre  ErrRng > 0
  post abs (RESULT ** 2-x) < ErrRng ;
```

Returns は、利子 interest で y 年間運用したとき、元利合計が何倍になるかを得る。

```
public static
  Returns : real +> int +> real
  Returns (interest)(y) ==
    (1 + interest) ** y
  pre  interest >= 0 and y > 0 ;
```

InterestOfImplicit は、years 年間運用して x 倍にしたいとき、必要な利子を得る関数の要求仕様。

```
  InterestOfImplicit : real * int +> real
  InterestOfImplicit (x, y) ==
    is not yet specified
   pre  x > 1 and y > 0 and ErrRng > 0
   post x > 1 and y > 0 and ErrRng > 0 and
         exists1 interest : real &
            let finalReturns = Returns (interest) (y) in
            abs (x-finalReturns) < ErrRng and RESULT = interest ;
```

Interest は、y 年間運用して x 倍にしたいとき、必要な利子を得る。

```
public static
   Interest : real +> int +> real
   Interest (x)(y) ==
     let f = lambda interest : real & x-Returns (interest) (y) in
     NewtonOfEquation (f) (x)
end
FReal
```

Test Suite : vdm.tc

Class : FReal

Name	#Calls	Coverage
FReal‘Root	1	√
FReal‘Round	8	√
FReal‘Figures	13	√
FReal‘Returns	57	√
FReal‘Interest	1	√
FReal‘Differentiate	18	√

Name	#Calls	Coverage
FReal‘FiguresOfInteger	17	√
FReal‘IsDecimalPlacesN	19	√
FReal‘NewtonOfEquation	2	√
FReal‘InterestOfImplicit	0	0%
FReal‘FiguresOfIntegerAux	26	√
FReal‘NumberOfDecimalPlaces	10	√
FReal‘NumberOfDecimalPlacesAux	23	√
Total Coverage		**82%**

2.23 FRealT

FReal のテストを行う。

```
class
FRealT
values
  e = 0
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 (),t3 (),t4 (),t5 ()] in
    FTestDriver`run (testcases);
```

2.23.1 「利子を得る Interest」を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FRealT01 : \t {「利子を得る Interest」を検査する",
         FReal`Interest (2) (10)-0.071773 < e);
```

2.23.2 「平方根を得る Root」を検査する

```
  t2 : () +> FTestDriver`TestCase
  t2 () ==
    mk_FTestDriver`TestCase
        (
         "FRealT02 : \t「平方根を得る Root」を検査する",
         let r = new FReal () in
         r.Root (2)-1.414214 < e);
```

2.23.3 「小数点以下 n 桁か？ tIsDecimalPlacesN」と「小数点以下何桁か？ NumberOfDecimalPlaces」を検査する

```
t3 : () +> FTestDriver`TestCase
t3() ==
  mk_FTestDriver`TestCase
      (
       "FRealT03 : \t「小数点以下 n 桁か？ tIsDecimalPlacesN」と「小数点以下何桁か
？ NumberOfDecimalPlaces」を検査する",
       let r = new FReal() in
       r.IsDecimalPlacesN(2)(10.01) and
       r.IsDecimalPlacesN(2)(-10.01) and
       r.IsDecimalPlacesN(3)(10.01) and
       r.IsDecimalPlacesN(3)(10.012) and
       r.IsDecimalPlacesN(3)(-10.012) and
       r.IsDecimalPlacesN(0)(10) and
       r.IsDecimalPlacesN(2)(10.011) = false and
       r.IsDecimalPlacesN(0)(10.1) = false and
       r.IsDecimalPlacesN(0)(-10.1) = false and
       r.NumberOfDecimalPlaces(-1.2) = 1 and
       r.NumberOfDecimalPlaces(1) = 0 and
       r.NumberOfDecimalPlaces(1) = 0 and
       r.NumberOfDecimalPlaces(1.23) = 2);
```

2.23.4 「桁数 Figures」を検査する

```
t4 : () +> FTestDriver`TestCase
t4() ==
  mk_FTestDriver`TestCase
      (
       "FRealT04 : \t「桁数 Figures」を検査する",
       let r = new FReal() in
       r.Figures(0) = 1 and
       r.Figures(1) = 1 and
       r.Figures(9) = 1 and
       r.Figures(10) = 2 and
       r.Figures(99) = 2 and
       r.Figures(100) = 3 and
       r.Figures(0.1) = 3 and
       r.Figures(9.1) = 3 and
       r.Figures(10.1) = 4 and
       r.Figures(10.123) = 6 and
       r.Figures(-10.123) = 6 and
       r.Figures(-0.1) = 3 and
       r.Figures(-0) = 1 and
       r.FiguresOfInteger(1) = 1 and
       r.FiguresOfInteger(-1) = 1 and
       r.FiguresOfInteger(10) = 2 and
       r.FiguresOfInteger(-10) = 2);
```

2.23.5 「小数点以下 n 桁で四捨五入する Round」を検査する

```
t5 : () +> FTestDriver`TestCase
t5() ==
  mk_FTestDriver`TestCase
      (
       "FRealT05 : \t「小数点以下 n 桁で四捨五入する Round」を検査する",
       FReal`Round(10.12345)(4) = 10.1235 and
       FReal`Round(10.12345)(3) = 10.123 and
       FReal`Round(10.12345)(2) = 10.12 and
       FReal`Round(10.125)(2) = 10.13 and
       FReal`Round(10.14)(1) = 10.1 and
       FReal`Round(10.15)(1) = 10.2 and
       FReal`Round(10.5)(0) = 11 and
       FReal`Round(10.4)(0) = 10)
```

```
end
FRealT
```

2.24 FSet

集合に関わる関数を提供する。集合演算であらかじめ定義された機能以外の機能を定義する。

```
class
FSet
```

AsSequence は、集合からシーケンスに変換する。

```
functions
public static
  AsSequence[@T] : set of @T -> seq of @T
  AsSequence(aSet) ==
    cases aSet :
      {} -> [],
      {x} union xs -> [x]^AsSequence[@T](xs)
    end
  post HasSameElems[@T](RESULT, aSet) ;
```

HasSameElems は、列が集合の要素を過不足無く含む事を表す。

```
public static
  HasSameElems[@T] : (seq of @T) * (set of @T) -> bool
  HasSameElems(s, aSet) ==
    (elems s = aSet) and (len s = card aSet);
```

Combinations は、集合から要素数 n 個の組み合わせを得る。

```
public static
  Combinations[@T] : nat1 +> set of @T +> set of set of @T
  Combinations(n)(aSet) ==
    {e|e in set power aSet & card e = n};
```

Fmap は、関数 f を集合に適用した結果の集合を返す。本関数は、通常の関数型言語の map 関数と同等であるが、map は VDM++ では予約語のため、Fmap と命名した。

```
public static
  Fmap[@T1, @T2] : (@T1 +> @T2) +> set of @T1 +> set of @T2
  Fmap(f)(aSet) ==
    {f(s)|s in set aSet}
end
FSet
```

Test Suite : vdm.tc

Class : FSet

Name	#Calls	Coverage
FSet‘Fmap	3	√
FSet‘AsSequence	11	√

Name	#Calls	Coverage
FSet‘Combinations	6	√
FSet‘HasSameElems	14	√
Total Coverage		**100%**

2.25 FSetT

FSet のテストを行う。testcase2 は、回帰テスト支援ライブラリのテストのため、意図的に false になるようにしている。

```
class
FSetT is subclass of FSet
functions
public static
  run : () -> bool
  run () ==
    let testcases = [t1 (),t2 (),t3 (),t4 ()] in
    FTestDriver`run (testcases);
```

2.25.1 HasSameElems を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FSetT01 : \t  Test HasSameElems",
         HasSameElems[int] (FSet`AsSequence[int] ({1,2,3,4}),{1,2,3,4}) and
         not HasSameElems[int] (FSet`AsSequence[int] ({1,2,3,4,5}),{1,2,3,4}) and
         not HasSameElems[int] ([1,2,3,4,4,5],{1,2,3,4}));
```

2.25.2 Combinations を検査する

```
  t2 : () +> FTestDriver`TestCase
  t2() ==
    mk_FTestDriver`TestCase
        (
         "FSetT02 : \t Test Combinations",
         Combinations[int](2)({1,2,3}) = {{1,2},{1,3},{2,3}} and
         Combinations[int](2)({1,2,3,4}) =
         {{1,2},{1,3},{1,4},{2,3},{2,4},{3,4}} and
         Fmap[set of int,set of set of int](Combinations[int](2))({{1,2,3},{1,2,3,4}}) =

         {{{1,2},{1,3},{2,3}},{{1,2},{1,3},{1,4},{2,3},{2,4},{3,4}}} and
         Combinations[int](3)({1,2,3,4}) = {{1,2,3},{1,2,4},{1,3,4},{2,3,4}} and
         Combinations[seq of char](2)({"Sahara","Sato","Sako","Yatsu","Nishikawa"}) =

         {{"Sahara","Sato"},{"Sahara","Nishikawa"},{"Sahara","Yatsu"},
          {"Sahara","Sako"},{"Sato","Nishikawa"},{"Sato","Yatsu"},
          {"Sato","Sako"},{"Nishikawa","Yatsu"},{"Nishikawa","Sako"},
          {"Yatsu","Sako"}});
```

2.25.3 Fmap を検査する

```
  t3 : () +> FTestDriver`TestCase
  t3() ==
    mk_FTestDriver`TestCase
        (
         "FSetT03 : \t Test Fmap",
         Fmap[int,int](lambda x : int & x mod 3)({1,2,3,4,5}) = {0,1,2});
```

2.25.4 Fmap のエラーケースを検査する

```
  t4 : () +> FTestDriver`TestCase
  t4() ==
    mk_FTestDriver`TestCase
        (
         "FSetT04 : \t Test error cases of Fmap",
         (FSet`Fmap[int,int](lambda x : int & x mod 3)({1,2,3,4,5}) = {0,1}) = false)
end
FSetT
```

2.26 FString

文字列（seq of char）に関わる関数を提供する。列型の関数を提供する FSequence クラスを継承し、列型で定義された機能以外の機能を定義する。

```
class
FString is subclass of FSequence
```

2.26.1 変換関数群

数字文字列を整数に変換する

```
functions
public static
   AsInteger : seq of char -> int
   AsInteger (s) ==
     FString`AsIntegerAux (s) (0)
   pre  IsDigits (s) ;
private static
   AsIntegerAux : seq of char +> int +> int
   AsIntegerAux (s)(sum) ==
     if s = []
     then sum
     else AsIntegerAux (tl s) (10 * sum + FCharacter`AsDigit (hd s));
```

2.26.2 判定関数群

数字文字列か判定する。

```
public static
   IsDigits : seq of char +> bool
   IsDigits (s) ==
     if s = []
     then true
     else FCharacter`IsDigit (hd s) and FString`IsDigits (tl s)
   pre  forall i in set inds s & FCharacter`IsDigit (s (i)) ;
```

空白かどうか判定する

```
public static
   IsSpace : [seq of char] +> bool
   IsSpace (s) ==
     if s = []
     then true
     else (hd s = ' ' or hd s = '\t' or hd s = '\n') and FString`IsSpace (tl s)
   post  forall i in set inds s & s (i) = ' ' or s (i) = '\t' or s (i) = '\n' ;
```

文字列の辞書順序での大小を判定する。

```
public static
   LT : seq of char +> seq of char +> bool
   LT(s1)(s2) ==
     cases mk_(s1,s2) :
       mk_([],[]) -> false,
       mk_([],-) -> true,
       mk_(- ^-,[]) -> false,
       mk_([x1]^xs1,[x2]^xs2) ->
           if FCharacter`LT(x1)(x2)
           then true
           elseif FCharacter`LT(x2)(x1)
           then false
           else FString`LT(xs1)(xs2)
     end;
public static
   LE : seq of char +> seq of char +> bool
   LE(s1)(s2) ==
     FString`LT(s1)(s2) or s1 = s2;
public static
   GT : seq of char +> seq of char +> bool
   GT(s1)(s2) ==
     FString`LT(s2)(s1);
public static
   GE : seq of char +> seq of char +> bool
   GE(s1)(s2) ==
     not FString`LT(s1)(s2);
```

Index は、指定された文字 c が文字列 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
   Index : char +> seq of char +> int
   Index(c)(s) ==
     FSequence`Index[char](c)(s);
```

IndexAll は、指定された文字 c が文字列 s の何番目にあるかを持つ自然数集合を返す。

```
public static
   IndexAll : char +> seq of char +> set of nat1
   IndexAll(c)(s) ==
     FSequence`IndexAll[char](c)(s);
```

文字列 s が、部分文字列として t を含むかを返す。

```
public static
```

```
  IsSubStr : seq of char +> seq of char +> bool
  IsSubStr(t)(s) ==
    if t = ""
    then true
    else let indexSet = IndexAll(t(1))(s) in
         exists i in set indexSet &
            SubStr(i)(len t)(s) = t;
```

2.26.3 文字列操作関数群

部分文字列を得る。

```
public static
  SubStr : nat +> nat +> seq of char +> seq of char
  SubStr(i)(numOfChars)(s) ==
    FSequence`SubSeq[char](i)(numOfChars)(s);
```

部分文字列を得る。ただし、文字列長が指定された文字数より小さいとき、指定された詰め文字を補充する。

```
public static
  SubStrFill : nat +> nat +> char +> seq of char +> seq of char
  SubStrFill(i)(numOfChars)(packChar)(s) ==
    let lastPos = i + numOfChars-1,
        appendLen = lastPos-len s in
    if appendLen <= 0
    then SubStr(i)(numOfChars)(s)
    else SubStr(i)(numOfChars)(s)^MkContChar(appendLen)(packChar);
```

指定された長さ（appendLen）の文字（packChar）から成る文字列を返す。

```
public static
  MkContChar : nat1 +> char +> seq of char
  MkContChar(appendLen)(packChar) ==
    let r = lambda x : seq of char & x^[packChar] in
    (r ** appendLen)("")
end
```

FString

Test Suite : vdm.tc

Class : FString

Name	#Calls	Coverage
FString`GE	0	0%
FString`GT	0	0%
FString`LE	1	66%
FString`LT	54	96%
FString`Index	226	√

Name	#Calls	Coverage
FString‘SubStr	57	√
FString‘IsSpace	0	0%
FString‘IndexAll	0	0%
FString‘IsDigits	0	0%
FString‘IsSubStr	0	0%
FString‘AsInteger	0	0%
FString‘MkContChar	0	0%
FString‘SubStrFill	0	0%
FString‘AsIntegerAux	0	0%
Total Coverage		**22%**

2.27 FStringT

文字列のテストを行う。

```
class
FStringT
functions
public static
  run : () -> bool
  run () ==
    let testcases =
            [
             t1 (),t2 (),t3 (),t4 (),t5 (),t6 ()] in
    FTestDriver`run (testcases);
```

2.27.1 文字列の大小を比較する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FStringT01 : \t 文字列の大小を比較する",
         let LT = FString`LT,
             LE = FString`LE,
             GT = FString`GT,
             GE = FString`GE in
         LT ("123") ("123") = false and
         GT ("123") ("123") = false and
         LE ("123") ("123") and
         LE ("123") ("1234") and
         GE ("123") ("1234") = false and
         not LE ("1234") ("123") and
         LE ("") ("") and
         FSequence`Fmap[seq of char,bool] (LT ("123")) (["123","1234","","223"])       =
[false,true,false,true] and
         FSequence`Fmap[seq of char,bool] (LE ("123")) (["1234",""]) = [true,false] and
         FSequence`Fmap[seq of char,bool] (GT ("123")) (["123","","23"]) = [false,true,false] and
         FSequence`Fmap[seq of char,bool] (GE ("123")) (["1234",""]) = [false,true]);
```

2.27.2 文字列が等しいかを比較する

```
t2 : () +> FTestDriver`TestCase
t2() ==
  mk_FTestDriver`TestCase
      (
       "FStringT02 : \t 文字列が等しいかを比較する",
       let s1234 = "1234",
           s = new FString() in
       s1234 = "1234" and
       s.IsSpace("") and
       s.IsSpace("  ") and
       s.IsSpace(" \t  ") and
       s.IsSpace(" \t  \n ") and
       s.IsSpace(" \t  \n a") = false and
       s.IsSpace([]));
```

2.27.3 部分文字列を得る

```
t3 : () +> FTestDriver`TestCase
t3() ==
  mk_FTestDriver`TestCase
      (
       "FStringT03 : \t 部分文字列を得る",
       let s = new FString(),
           SubStr = FString`SubStr in
       s.SubStr(6)(6)("Shin Sahara") = "Sahara" and
       s.SubStr(6)(8)("Shin Sahara") = "Sahara" and
       s.SubStr(6)(3)("Shin Sahara") = "Sah" and
       s.SubStr(1)(0)("Shin Sahara") = "" and
       s.SubStrFill(1)(3)('*')("sahara") = "sah" and
       s.SubStrFill(1)(6)('*')("sahara") = "sahara" and
       s.SubStrFill(1)(10)('*')("sahara") = "sahara****" and
       s.SubStrFill(3)(4)('*')("sahara") = "hara" and
       s.SubStrFill(3)(10)('*')("sahara") = "hara******" and
       s.SubStrFill(1)(0)('*')("sahara") = "" and
       s.SubStrFill(1)(6)('*')("") = "******" and
       FString`SubStr(6)(6)("Shin Sahara") = "Sahara" and
       SubStr(6)(8)("Shin Sahara") = "Sahara" and
       FSequence`Fmap[seq of char, seq of char](SubStr(6)(8))(["1234567890", "12345671"]) =
["67890", "671"]);
```

2.27.4 数字文字列の扱いを検査する

```
t4 : () +> FTestDriver`TestCase
t4() ==
  mk_FTestDriver`TestCase
      (
       "FStringT04 : \t 数字文字列の扱いを検査する",
       FString`IsDigits("1234567890") = true and
       FString`IsDigits("abc") = false and
       FString`AsInteger("1234567890") = 1234567890 and
       FString`AsInteger("") = 0);
```

2.27.5 指定した文字が、文字列に最初に出現する位置を検査する

```
t5 : () +> FTestDriver`TestCase
t5() ==
  mk_FTestDriver`TestCase
      (
       "FStringT05 : \t 指定した文字が、文字列に最初に出現する位置を検査する",
       FString`Index('1')("1234567890") = 1 and
       FString`Index('0')("1234567890") = 10 and
       FString`Index('a')("1234567890") = 0 and
       FString`IndexAll('1')("1234567890") = {1} and
       FString`IndexAll('0')("1234567890") = {10} and
       FString`IndexAll('a')("1234567890") = {} and
       FString`IndexAll('1')("1231567190") = {1,4,8} and
       FString`IndexAll('1')("1231567191") = {1,4,8,10} and
       FString`Index('1')("1234567890") = 1 and
       FString`Index('0')("1234567890") = 10 and
       FString`Index('a')("1234567890") = 0 and
       FString`IndexAll('1')("1234567890") = {1} and
       FString`IndexAll('0')("1234567890") = {10} and
       FString`IndexAll('a')("1234567890") = {} and
       FString`IndexAll('1')("1231567190") = {1,4,8} and
       FString`IndexAll('1')("1231567191") = {1,4,8,10} and
       FSequence`Fmap[seq of char,int](FString`Index('1'))(["1234567890","2345671"]) =
[1,7] and
       FSequence`Fmap[seq of char,set of int](FString`IndexAll('1'))(["1231567190","1231567191"]) =
[{1,4,8},{1,4,8,10}]);
```

2.27.6 指定した文字列が、ある文字列に含まれるかを検査する

```
  t6 : () +> FTestDriver`TestCase
  t6() ==
    mk_FTestDriver`TestCase
        (
         "FStringT06 : \t 指定した文字列が、ある文字列に含まれるかを検査する",
         let IsSubStr = FString`IsSubStr in
         FString`IsSubStr("abc")("1234567890") = false and
         IsSubStr("Sahara")("Sahara") = true and
         IsSubStr("Saha")("Sahara") = true and
         IsSubStr("hara")("Sahara") = true and
         IsSubStr("ra")("Sahara") = true and
         IsSubStr("")("Sahara") = true)
end
FStringT
```

2.28 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。

2.28.1 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
  Sum[@T] : seq of @T +> @T
  Sum(s) ==
    Foldl[@T,@T](Plus[@T])(0)(s)
  pre  is_(s,seq of int) or is_(s,seq of nat) or is_(s,seq of nat1) or
       is_(s,seq of real) or
       is_(s,seq of rat) ;
```

Prod は列 s の全要素の積を返す。

```
public static
  Prod[@T] : seq of @T +> @T
  Prod(s) ==
    Foldl[@T,@T](Product[@T])(1)(s)
  pre  is_(s,seq of int) or is_(s,seq of nat) or is_(s,seq of nat1) or
       is_(s,seq of real) or
       is_(s,seq of rat) ;
```

Plus は加算を行う。

```
public static
  Plus[@T] : @T +> @T +> @T
  Plus(a)(b) ==
    a + b;
```

Product は積算を行う。

```
public static
  Product[@T] : @T +> @T +> @T
  Product(a)(b) ==
    a * b;
```

Append は列の追加を行う。

```
public static
```

```
  Append[@T] : seq of @T +> @T +> seq of @T
  Append(s)(e) ==
    s^[e];
```

Average は列 s の要素の平均を求める。

```
public static
  Average[@T] : seq of @T +> [real]
  Average(s) ==
    if s = []
    then nil
    else AverageAux[@T](0)(0)(s)
  post if s = []
       then RESULT = nil
       else RESULT = Sum[@T](s)/len s ;

  AverageAux[@T] : @T +> @T +> seq of @T +> real
  AverageAux(total)(numOfElem)(s) ==
    cases s :
      [x]^xs -> AverageAux[@T](total + x)(numOfElem + 1)(xs),
      [] -> total/numOfElem
    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
  IsAscendingInTotalOrder[@T] : (@T * @T +> bool) +> seq of @T +> bool
  IsAscendingInTotalOrder(f)(s) ==
    forall i,j in set inds s & i < j => f(s(i),s(j)) or 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
  IsDescendingInTotalOrder[@T] : (@T * @T +> bool) +> seq of @T +> bool
  IsDescendingInTotalOrder(f)(s) ==
    forall i,j in set inds s & i < j => f(s(j),s(i)) or s(i) = s(j);
```

IsAscending は、演算子¡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
  IsAscending[@T] : seq of @T +> bool
  IsAscending(s) ==
    IsAscendingInTotalOrder[@T](lambda x : @T, y : @T & x < y)(s);
```

IsDescending は、演算子¡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
```

```
  IsDescending[@T] : seq of @T +> bool
  IsDescending(s) ==
    IsDescendingInTotalOrder[@T](lambda 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
  Sort[@T] : (@T * @T +> bool) +> seq of @T +> seq of @T
  Sort(f)(s) ==
    cases s :
      [] -> [],
      [x]^xs ->
          Sort[@T](f)([xs(i)|i in set inds xs & f(xs(i),x)])^
          [x]^
          Sort[@T](f)([xs(i)|i in set inds xs & not f(xs(i),x)])
    end;
```

AscendingSort は、演算子¡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
  AscendingSort[@T] : seq of @T +> seq of @T
  AscendingSort(s) ==
    Sort[@T](lambda x : @T, y : @T & x < y)(s)
  post IsAscending[@T](RESULT) ;
```

DescendingSort は、演算子¿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¡から見れば降順。

```
public static
  DescendingSort[@T] : seq of @T +> seq of @T
  DescendingSort(s) ==
    Sort[@T](lambda 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
```

```
  IsOrdered[@T] : seq of (@T * @T +> bool) +> seq of @T +> seq of @T +> bool
  IsOrdered(f)(s1)(s2) ==
    cases mk_(s1,s2) :
      mk_([],[]) -> false,
      mk_([],-) -> true,
      mk_(-,[]) -> false,
      mk_([x1]^xs1,[x2]^xs2) ->
          if (hd f)(x1,x2)
          then true
          elseif (hd f)(x2,x1)
          then false
          else IsOrdered[@T](tl f)(xs1)(xs2)
    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
  Merge[@T] : (@T * @T +> bool) +> seq of @T +> seq of @T +> seq of @T
  Merge(f)(s1)(s2) ==
    cases mk_(s1,s2) :
      mk_([],y) -> y,
      mk_(x,[]) -> x,
      mk_([x1]^xs1,[x2]^xs2) ->
          if f(x1,x2)
          then [x1]^FSequence`Merge[@T](f)(xs1)(s2)
          else [x2]^FSequence`Merge[@T](f)(s1)(xs2)
    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
  InsertAt[@T] : nat1 +> @T +> seq of @T +> seq of @T
  InsertAt(i)(e)(s) ==
    cases mk_(i,s) :
      mk_(1,s1) -> [e]^s1,
      mk_(-,[]) -> [e],
      mk_(i1,[x]^xs) -> [x]^InsertAt[@T](i1-1)(e)(xs)
    end;
```

RemoveAt は、列 s の指定された位置 i の要素を削除する。

```
public static
```

```
  RemoveAt[@T] : nat1 +> seq of @T +> seq of @T
  RemoveAt(i)(s) ==
    cases mk_(i,s) :
      mk_(1,[-]^xs) -> xs,
      mk_(i1,[x]^xs) -> [x]^RemoveAt[@T](i1-1)(xs),
      mk_(-,[]) -> []
    end;
```

RemoveDup は、列 s から重複する要素を削除する。

```
public static
  RemoveDup[@T] : seq of @T +> seq of @T
  RemoveDup(s) ==
    cases s :
      [x]^xs -> [x]^RemoveDup[@T](Filter[@T](lambda e : @T & e <> x)(xs)),
      [] -> []
    end
  post not IsDup[@T](RESULT) ;
```

RemoveMember は、列 s から要素 e を削除する。

```
public static
  RemoveMember[@T] : @T +> seq of @T +> seq of @T
  RemoveMember(e)(s) ==
    cases s :
      [x]^xs -> if e = x
                then xs
                else [x]^RemoveMember[@T](e)(xs),
      [] -> []
    end;
```

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

```
public static
  RemoveMembers[@T] : seq of @T +> seq of @T +> seq of @T
  RemoveMembers(es)(s) ==
    cases es :
      [] -> s,
      [x]^xs -> RemoveMembers[@T](xs)(RemoveMember[@T](x)(s))
    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
```

```
  UpdateAt[@T] : nat1 +> @T +> seq of @T +> seq of @T
  UpdateAt(i)(e)(s) ==
    cases mk_(i,s) :
      mk_(-,[]) -> [],
      mk_(1,[-]^xs) -> [e]^xs,
      mk_(i1,[x]^xs) -> [x]^UpdateAt[@T](i1-1)(e)(xs)
    end;
```

Take は、列 s の先頭 i 個からなる列を返す。

```
public static
  Take[@T] : int +> seq of @T +> seq of @T
  Take(i)(s) ==
    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
  TakeWhile[@T] : (@T +> bool) +> seq of @T +> seq of @T
  TakeWhile(f)(s) ==
    cases s :
      [x]^xs ->
          if f(x)
          then [x]^TakeWhile[@T](f)(xs)
          else [],
      [] -> []
    end;
```

Drop は、列 s の先頭 i 個を除く列を返す。

```
public static
  Drop[@T] : int +> seq of @T +> seq of @T
  Drop(i)(s) ==
    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
  DropWhile[@T] : (@T +> bool) +> seq of @T +> seq of @T
  DropWhile(f)(s) ==
    cases s :
      [x]^xs ->
          if f(x)
          then DropWhile[@T](f)(xs)
          else s,
      [] -> []
    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って以降の列の組に分ける。

```
public static
  Span[@T] : (@T +> bool) +> seq of @T +> seq of @T * seq of @T
  Span(f)(s) ==
    cases s :
      [x]^xs ->
          if f(x)
          then let mk_(satisfied, notSatisfied) = Span[@T](f)(xs) in
               mk_([x]^satisfied, notSatisfied)
          else mk_([], s),
      [] -> mk_([], [])
    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
  SubSeq[@T] : nat +> nat +> seq of @T +> seq of @T
  SubSeq(i)(numOfElems)(s) ==
    s(i, ..., i + numOfElems-1);
```

Last は、列 s の最後の要素を返す。

```
public static
  Last[@T] : seq of @T +> @T
  Last(s) ==
    s(len s);
```

Fmap は、関数を列に適用した結果の列を返す。

```
public static
  Fmap[@T1, @T2] : (@T1 +> @T2) +> seq of @T1 +> seq of @T2
  Fmap(f)(s) ==
    [f(s(i))|i in set 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
  Filter[@T] : (@T +> bool) +> seq of @T +> seq of @T
  Filter(f)(s) ==
    [s(i)|i in set 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
```

```
  Foldl[@T1,@T2] : (@T1 +> @T2 +> @T1) +> @T1 +> seq of @T2 +> @T1
  Foldl(f)(args)(s) ==
    cases s :
      [] -> args,
      [x]^xs -> Foldl[@T1,@T2](f)(f(args)(x))(xs)
    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
  Foldr[@T1,@T2] : (@T1 +> @T2 +> @T2) +> @T2 +> seq of @T1 +> @T2
  Foldr(f)(args)(s) ==
    cases s :
      [] -> args,
      [x]^xs -> f(x)(Foldr[@T1,@T2](f)(args)(xs))
    end;
```

IsMember は、要素があるか否かを返す。

```
public static
  IsMember[@T] : @T +> seq of @T +> bool
  IsMember(e)(s) ==
    cases s :
      [x]^xs -> e = x or IsMember[@T](e)(xs),
      [] -> false
    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
  IsAnyMember[@T] : seq of @T +> seq of @T +> bool
  IsAnyMember(es)(s) ==
    cases es :
      [x]^xs -> IsMember[@T](x)(s) or IsAnyMember[@T](xs)(s),
      [] -> false
    end;
```

IsDup は、列 s 中に同じ要素があるか否かを返す。

```
public static
  IsDup[@T] : seq of @T +> bool
  IsDup(s) ==
    not card elems s = len s
  post if s = []
       then RESULT = false
       else RESULT = not  forall i,j in set inds s & s(i) <> s(j) <=> i <> 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
   Index[@T] : @T +> seq of @T +> int
   Index(e)(s) ==
     let i = 0 in
     IndexAux[@T](e)(s)(i);
public static
   IndexAux[@T] : @T +> seq of @T +> int +> int
   IndexAux(e)(s)(i) ==
     cases s :
       [] -> 0,
       [x]^xs ->
           if x = e
           then i + 1
           else IndexAux[@T](e)(xs)(i + 1)
     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
   IndexAll[@T] : @T +> seq of @T +> set of nat1
   IndexAll(e)(s) ==
     {i|i in set 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
   Flatten[@T] : seq of seq of @T +> seq of @T
   Flatten(s) ==
     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
   Compact[@T] : seq of [@T] +> seq of @T
   Compact(s) ==
     [s(i)|i in set inds s & s(i) <> nil]
   post  forall i in set inds RESULT & RESULT(i) <> 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
   Freverse[@T] : seq of @T +> seq of @T
   Freverse(s) ==
     [s(len s + 1-i)|i in set inds s];
```

Permutations は、列 s から順列を得る

```
public static
```

```
  Permutations[@T] : seq of @T +> set of seq of @T
  Permutations (s) ==
    cases s :
      [], [−] -> {s},
      others -> dunion {{[s (i)]^j|j in set Permutations[@T] (RemoveAt[@T] (i) (s))}|i in set inds s}
    end
  post  forall x in set RESULT & elems x = 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
  IsPermutations[@T] : seq of @T +> seq of @T +> bool
  IsPermutations (s)(t) ==
    RemoveMembers[@T] (s) (t) = [] and RemoveMembers[@T] (t) (s) = [];
```

Unzip は、組の列を、列の組に変換する

```
public static
  Unzip[@T1, @T2] : seq of (@T1 * @T2) +> seq of @T1 * seq of @T2
  Unzip (s) ==
    cases s :
      [] -> mk_ ([], []),
      [mk_ (x, y)]^xs ->
          let mk_ (s1, t) = Unzip[@T1, @T2] (xs) in
          mk_ ([x]^s1, [y]^t)
    end;
```

Zip は、列の組を、組の列に変換する

```
public static
  Zip[@T1, @T2] : seq of @T1 * seq of @T2 +> seq of (@T1 * @T2)
  Zip (s1, s2) ==
    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
  Zip2[@T1, @T2] : seq of @T1 +> seq of @T2 +> seq of (@T1 * @T2)
  Zip2 (s1)(s2) ==
    cases mk_ (s1, s2) :
      mk_ ([x1]^xs1, [x2]^xs2) -> [mk_ (x1, x2)]^Zip2[@T1, @T2] (xs1) (xs2),
      mk_ (−, −) -> []
    end
end
```

FSequence

Test Suite : vdm.tc

Class : FSequence

Name	#Calls	Coverage
FSequence‘Sum	6	√
FSequence‘Zip	4	√
FSequence‘Drop	2	√
FSequence‘Fmap	7	√
FSequence‘Last	1	√
FSequence‘Plus	31	√
FSequence‘Prod	5	√
FSequence‘Sort	35	√
FSequence‘Span	14	√
FSequence‘Take	9	√
FSequence‘Zip2	16	√
FSequence‘Foldl	57	√
FSequence‘Foldr	12	√
FSequence‘Index	867	√
FSequence‘IsDup	15	√
FSequence‘Merge	16	√
FSequence‘Unzip	5	√
FSequence‘Append	3	√
FSequence‘Filter	9	√
FSequence‘SubSeq	58	√
FSequence‘Average	3	√
FSequence‘Compact	3	√
FSequence‘Flatten	1	√
FSequence‘Product	15	√
FSequence‘Freverse	21	√
FSequence‘IndexAll	4	√
FSequence‘IndexAux	4911	√
FSequence‘InsertAt	20	√
FSequence‘IsMember	39	√
FSequence‘RemoveAt	62	√
FSequence‘UpdateAt	22	√
FSequence‘DropWhile	6	√
FSequence‘IsOrdered	15	√
FSequence‘RemoveDup	11	√
FSequence‘TakeWhile	6	√
FSequence‘AverageAux	9	√
FSequence‘IsAnyMember	6	√

Name	#Calls	Coverage
FSequence‘IsAscending	3	√
FSequence‘IsDescending	2	√
FSequence‘Permutations	24	√
FSequence‘RemoveMember	86	√
FSequence‘AscendingSort	1	√
FSequence‘RemoveMembers	62	√
FSequence‘DescendingSort	1	√
FSequence‘IsPermutations	6	√
FSequence‘IsAscendingInTotalOrder	3	√
FSequence‘IsDescendingInTotalOrder	4	√
Total Coverage		**100%**

2.29 FSequenceT

FSequence のテストを行う。

```
class
FSequenceT
types
  public TestType = int|seq of char|char;
public
  Record::v : int
          str : seq of char
          c : char
functions
public static
  run : () -> bool
  run () ==
    let testcases =
            [
             t1 (),t2 (),t3 (),t4 (),t5 (),t6 (),t7 (),t8 (),t9 (),t10 (),
             t11 (),t12 (),t13 (),t14 (),t15 (),t16 (),t17 (),t18 (),t19 (),t20 (),
             t21 (),t22 (),t23 (),t24 ()] in
    FTestDriver`run (testcases);
```

2.29.1 合計と積を検査する

```
  t1 : () +> FTestDriver`TestCase
  t1 () ==
    mk_FTestDriver`TestCase
        (
         "FSequenceT01 : \t 合計と積を検査する",
         FSequence`Sum[int] ([1, 2, 3, 4, 5, 6, 7, 8, 9]) = 45 and
         FSequence`Sum[int] ([]) = 0 and
         FSequence`Prod[int] ([2, 3, 4]) = 24 and
         FSequence`Prod[int] ([]) = 1 and
         FSequence`Sum[real] ([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]) = 4.5 and
         FSequence`Sum[real] ([]) = 0 and
         FSequence`Prod[real] ([2, 3, 4]) = 24 and
         FSequence`Prod[real] ([]) = 1 and
         FSequence`Prod[real] ([2.1, 3.2, 4.3]) = 2.1 * 3.2 * 4.3);
```

2.29.2 全順序昇順か？を検査する

```
t2 : () +> FTestDriver`TestCase
t2() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT02 : \t 全順序昇順か？を検査する",
       FSequence`IsAscending[int]([1,2,4,4,7,8,8,8]) and
       not FSequence`IsAscending[real]([1,2,3,1.5]));
```

2.29.3 全順序降順か？を検査する

```
t3 : () +> FTestDriver`TestCase
t3() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT03 : \t 全順序降順か？を検査する",
       FSequence`IsDescending[int]([3,2,2,1,1]) and
       FSequence`IsDescendingInTotalOrder[int](lambda x : int, y : int & x < y)([3,2,2,1,1]) and
       FSequence`IsDescendingInTotalOrder[int](lambda x : int, y : int & x < y)([3,2,2,1,2]) =
false);
```

2.29.4 順番になっているか？を検査する

```
t4 : () +> FTestDriver`TestCase
t4 () ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT04 : \t 順番になっているか？を検査する",
       let sq = new FSequence (),
           fs =
               [(lambda x : int, y : int & x < y),
                lambda x : seq of char, y : seq of char & FString`LT (x) (y),
                lambda x : char, y : char & FCharacter`LT (x) (y)],
           f =
               lambda x : Record, y : Record &
                   FSequence`IsOrdered[TestType]                          (
       FSequence`Sort[Record] (f)
           (
            [mk_Record (10, "sahara",'c'), mk_Record (10, "sahara",'a')]) =
       [mk_Record (10, "sahara",'a'), mk_Record (10, "sahara",'c')] and
       sq.IsOrdered[TestType] (fs) ([3, "123",'a']) ([3, "123",'A']) = true and
       sq.IsOrdered[TestType] (fs) ([3, "123",'a']) ([3, "123",'0']) = false and
       sq.IsOrdered[int|seq of char|char] (fs) ([]) ([]) = false and
       sq.IsOrdered[int|seq of char|char] (fs) ([]) ([3, "123",'0']) = true and
       sq.IsOrdered[int|seq of char|char] (fs) ([3, "123",'0']) ([]) = false);
```

2.29.5 マージを検査する

```
t5 : () +> FTestDriver`TestCase
t5 () ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT05 : \t マージを検査する",
       let sq = new FSequence (),
           f1 = lambda x : int, y : int & x < y,
           f2 = lambda x : char, y : char & FCharacter`LT (x) (y) in
       sq.Merge[int] (f1) ([1, 4, 6]) ([2, 3, 4, 5]) = [1, 2, 3, 4, 4, 5, 6] and
       sq.Merge[char] (f2) ("146") ("2345") = "1234456" and
       sq.Merge[char] (f2) ("") ("2345") = "2345" and
       sq.Merge[char] (f2) ("146") ("") = "146");
```

2.29.6 文字列操作を検査する

```
t6 : () +> FTestDriver`TestCase
t6() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT06 : \t 文字列操作を検査する",
       let sq = new FSequence() in
       sq.Take[int](2)([2,3,4,5]) = [2,3] and
       sq.Drop[char](5)("Shin Sahara") = "Sahara" and
       sq.Last[int]([1,2,3]) = 3 and
       sq.Filter[int](lambda x : int & x mod 2 = 0)([1,2,3,4,5,6]) = [2,4,6] and
       FSequence`SubSeq[char](4)(3)("1234567890") = "456" and
       FSequence`Flatten[int]([[1,2,3],[3,4],[4,5,6]]) = [1,2,3,3,4,4,5,6]);
```

2.29.7 ソートを検査する

```
t7 : () +> FTestDriver`TestCase
t7() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT07 : \t ソートを検査する",
       FSequence`AscendingSort[int]([3,1,6,4,2,6,5]) = [1,2,3,4,5,6,6] and
       FSequence`DescendingSort[int]([3,1,6,4,2,6,5]) = [6,6,5,4,3,2,1]);
```

2.29.8 空要素削除を検査する

```
t8 : () +> FTestDriver`TestCase
t8() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT08 : \t 空要素削除を検査する",
       FSequence`Compact[[int]]([3,1,6,4,nil,2,6,5,nil]) = [3,1,6,4,2,6,5] and
       FSequence`Compact[[int]]([nil,nil]) = [] and
       FSequence`Compact[[int]]([]) = []);
```

2.29.9 列の反転を検査する

```
t9 : () +> FTestDriver`TestCase
t9() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT09 : \t 列の反転を検査する",
       FSequence`Freverse[[int]]([3,1,6,4,nil,2,6,5,nil]) = [nil,5,6,2,nil,4,6,1,3] and
       FSequence`Freverse[[int]]([]) = []);
```

2.29.10 順列を検査する

```
t10 : () +> FTestDriver`TestCase
t10() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT10 : \t 順列を検査する",
       FSequence`Permutations[[int]]([1,2,3]) =
       {[1,2,3],
        [1,3,2],
        [2,1,3],
        [2,3,1],
        [3,1,2],
        [3,2,1]} and
       FSequence`Permutations[[int]]([1,2,2]) =
       {[1,2,2],
        [2,1,2],
        [2,2,1]} and
       FSequence`Permutations[[bool]]([true,false]) =
       {[true,false],
        [false,true]} and
       FSequence`Permutations[[int]]([]) = {[]} and
       FSequence`IsPermutations[int]([1,2,3])([1,3,2]) and
       FSequence`IsPermutations[int]([1,2,3])([2,1,3]) and
       FSequence`IsPermutations[int]([1,2,3])([2,3,1]) and
       FSequence`IsPermutations[int]([1,2,3])([3,1,2]) and
       FSequence`IsPermutations[int]([1,2,3])([3,2,1]) and
       FSequence`IsPermutations[int]([1,2,3])([3,2,2]) = false);
```

2.29.11 列の要素か？を検査する

```
t11 : () +> FTestDriver`TestCase
t11() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT11 : \t 列の要素か？を検査する",
       FSequence`IsMember[int](2)([1,2,3,4,5,6]) and
       FSequence`IsMember[int](0)([1,2,3,4,5,6]) = false and
       FSequence`IsMember[int](6)([1,2,3,4,5,6]) and
       FSequence`IsAnyMember[int]([6])([1,2,3,4,5,6]) and
       FSequence`IsAnyMember[int]([0,7])([1,2,3,4,5,6]) = false and
       FSequence`IsAnyMember[int]([4,6])([1,2,3,4,5,6]) and
       FSequence`IsAnyMember[int]([])([1,2,3,4,5,6]) = false);
```

2.29.12 Fmap を検査する

```
t12 : () +> FTestDriver`TestCase
t12() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT12 : \t Fmap を検査する",
       FSequence`Fmap[int,int](lambda x : int & x mod 3)([1,2,3,4,5]) = [1,2,0,1,2] and
       FSequence`Fmap[seq of char,seq of char]
           (
            FSequence`Take[char](2))(["Sahara","Sakoh"]) = ["Sa","Sa"]);
```

2.29.13 Index, IndexAll を検査する

```
t13 : () +> FTestDriver`TestCase
t13() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT13 : \t Index, IndexAll を検査する",
       let index = FSequence`Index,
           indexAll = FSequence`IndexAll in
       index[int](1)([1,2,3,4,5]) = 1 and
       index[int](5)([1,2,3,4,5]) = 5 and
       index[int](9)([1,2,3,4,5]) = 0 and
       index[char]('b')(['a','b','c']) = 2 and
       index[char]('z')(['a','b','c']) = 0 and
       indexAll[int](9)([1,2,3,4,5]) = {} and
       indexAll[int](9)([]) = {} and
       indexAll[int](1)([1,2,3,4,1]) = {1,5} and
       indexAll[int](1)([1,2,3,4,1,1]) = {1,5,6});
```

2.29.14 Average を検査する

```
t14 : () +> FTestDriver`TestCase
t14() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT14 : \t Average を検査する",
       let avg1 = FSequence`Average[int],
           avg2 = FSequence`Average[real] in
       avg1([]) = nil and
       avg1([1,2,3,4]) = (1+2+3+4)/4 and
       avg2([1.3,2.4,3.5]) = 7.2/3);
```

2.29.15 挿入を検査する

```
t15 : () +> FTestDriver`TestCase
t15() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT15 : \t 挿入を検査する",
       let ins1 = FSequence`InsertAt[int],
           ins2 = FSequence`InsertAt[char] in
       ins1(1)(1)([2,3,4,5]) = [1,2,3,4,5] and
       ins1(3)(3)([1,2,4,5]) = [1,2,3,4,5] and
       ins1(3)(3)([1,2]) = [1,2,3] and
       ins1(4)(3)([1,2]) = [1,2,3] and
       ins1(5)(3)([1,2]) = [1,2,3] and
       ins2(1)('1')("2345") = "12345" and
       ins2(3)('3')("1245") = "12345" and
       ins2(3)('3')("12") = "123");
```

2.29.16 削除を検査する

```
t16 : () +> FTestDriver`TestCase
t16() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT16 : \t 削除を検査する",
       let rm1 = FSequence`RemoveAt[int],
           rm2 = FSequence`RemoveAt[char] in
       rm1(1)([1,2,3,4,5]) = [2,3,4,5] and
       rm1(3)([1,2,4,3]) = [1,2,3] and
       rm1(3)([1,2]) = [1,2] and
       rm1(4)([1,2]) = [1,2] and
       rm1(5)([1,2]) = [1,2] and
       rm2(1)("12345") = "2345" and
       rm2(3)("1243") = "123" and
       rm2(3)("12") = "12");
```

2.29.17 更新を検査する

```
t17 : () +> FTestDriver`TestCase
t17() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT17 : \t 更新を検査する",
       let up1 = FSequence`UpdateAt[int],
           up2 = FSequence`UpdateAt[char] in
       up1(1)(10)([1,2,3,4,5]) = [10,2,3,4,5] and
       up1(3)(40)([1,2,4,3]) = [1,2,40,3] and
       up1(2)(30)([1,2]) = [1,30] and
       up1(3)(30)([1,2]) = [1,2] and
       up1(4)(30)([1,2]) = [1,2] and
       up2(1)('a')("12345") = "a2345" and
       up2(3)('b')("1243") = "12b3" and
       up2(3)('c')("123") = "12c" and
       up2(3)('c')("12") = "12");
```

2.29.18 複数削除を検査する

```
t18 : () +> FTestDriver`TestCase
t18() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT18 : \t 複数削除を検査する",
       let removeDup = FSequence`RemoveDup[int],
           removeMember = FSequence`RemoveMember[int],
           removeMembers = FSequence`RemoveMembers[int] in
       removeDup([]) = [] and
       removeDup([1,1,2,2,2,3,4,4,4,4]) = [1,2,3,4] and
       removeDup([1,2,3,4]) = [1,2,3,4] and
       removeMember(1)([]) = [] and
       removeMember(1)([1,2,3]) = [2,3] and
       removeMember(4)([1,2,3]) = [1,2,3] and
       removeMembers([])([]) = [] and
       removeMembers([])([1,2,3]) = [1,2,3] and
       removeMembers([1,2,3])([]) = [] and
       removeMembers([1,2,3])([1,2,3]) = [] and
       removeMembers([1,4,5])([1,2,3,4]) = [2,3] and
       removeMembers([1,4,5])([1,2,3,4,1,2,3,4,1]) = [2,3,1,2,3,4,1]);
```

2.29.19 zip を検査する

```
t19 : () +> FTestDriver`TestCase
t19() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT19 : \t zip を検査する",
       let zip = FSequence`Zip[int, char],
           zip2 = FSequence`Zip2[int, char],
           unzip = FSequence`Unzip[int, char] in
       zip([], []) = [] and
       zip([1, 2, 3], ['a','b','c']) = [mk_(1,'a'), mk_(2,'b'), mk_(3,'c')] and
       zip([1, 2], ['a','b','c']) = [mk_(1,'a'), mk_(2,'b')] and
       zip([1, 2, 3], ['a','b']) = [mk_(1,'a'), mk_(2,'b')] and
       zip2([])([]) = [] and
       zip2([1, 2, 3])(['a','b','c']) = [mk_(1,'a'), mk_(2,'b'), mk_(3,'c')] and
       unzip([]) = mk_([], []) and
       unzip([mk_(1,'a'), mk_(2,'b'), mk_(3,'c')]) = mk_([1, 2, 3], ['a','b','c']));
```

2.29.20 Span を検査する

```
t20 : () +> FTestDriver`TestCase
t20() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT20 : \t Span を検査する",
       let span = FSequence`Span[int],
           p1 = lambda x : int & x mod 2 = 0,
           p2 = lambda x : int & x < 10 in
       span(p1)([]) = mk_([], []) and
       span(p1)([2, 4, 6, 1, 3]) = mk_([2, 4, 6], [1, 3]) and
       span(p2)([1, 2, 3, 4, 5]) = mk_([1, 2, 3, 4, 5], []) and
       span(p2)([1, 2, 12, 13, 4, 15]) = mk_([1, 2], [12, 13, 4, 15]));
```

2.29.21 TakeWhile, DropWhile を検査する

```
t21 : () +> FTestDriver`TestCase
t21() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT21 : \t TakeWhile, DropWhile を検査する",
       let TakeWhile = FSequence`TakeWhile[int],
           DropWhile = FSequence`DropWhile[int],
           p1 = lambda x : int & x mod 2 = 0 in
       TakeWhile(p1)([]) = [] and
       TakeWhile(p1)([2,4,6,8,1,3,5,2,4]) = [2,4,6,8] and
       DropWhile(p1)([]) = [] and
       DropWhile(p1)([2,4,6,8,1,2,3,4,5]) = [1,2,3,4,5]);
```

2.29.22 Foldl を検査する

```
t22 : () +> FTestDriver`TestCase
t22() ==
  mk_FTestDriver`TestCase
      (
       "FSequenceT22 : \t Foldl を検査する",
       let foldl = FSequence`Foldl[int,int],
           f2 = FSequence`Foldl[seq of char,char],
           plus = FSequence`Plus[int],
           prod = FSequence`Product[int] in
       foldl(plus)(0)([1,2,3]) = 6 and
       foldl(prod)(1)([2,3,4]) = 24 and
       f2(FSequence`Append[char])([])("abc") = "abc");
```

2.29.23 Foldr を検査する

```
  t23 : () +> FTestDriver`TestCase
  t23() ==
    mk_FTestDriver`TestCase
        (
         "FSequenceT23 : \t Foldr を検査する",
         let removeAt = FSequence`RemoveAt[char],
             foldr = FSequence`Foldr[int, int],
             f3 = FSequence`Foldr[nat1, seq of char],
             plus = FSequence`Plus[int],
             prod = FSequence`Product[int] in
         foldr(plus)(0)([1, 2, 3]) = 6 and
         foldr(prod)(1)([2, 3, 4]) = 24 and
         f3(removeAt)("12345")([1, 3, 5]) = "24");
```

2.29.24 IsDup を検査する

```
  t24 : () +> FTestDriver`TestCase
  t24() ==
    mk_FTestDriver`TestCase
        (
         "FSequenceT24 : \t IsDup を検査する",
         FSequence`IsDup[int]([1, 2, 3]) = false and
         FSequence`IsDup[int]([1, 2, 2, 3]) and
         FSequence`IsDup[int]([]) = false and
         FSequence`IsDup[int]([1, 2, 3, 1]))
end
FSequenceT
```

2.30 FTestDriver

回帰テストを実行するモジュール。

TestCase 型は、テストケース 1 件を表す。

```
class
FTestDriver
types
public
   TestCase::testCaseName : seq of char
             testResult : bool
```

run は、与えられたテストケース列から結果列を得る。結果がすべて true ならば全体成功メッセージを表示し、1 つでも失敗があれば全体失敗メッセージを表示する。

```
functions
public static
   run : seq of TestCase -> bool
   run (t) ==
     let m = "Result - of - testcases.",
         r = [isOK (t (i))|i in set inds t] in
     if  forall i in set inds r & r (i)
     then FTestLogger`SuccessAll (m)
     else FTestLogger`FailureAll (m);
```

isOK は、与えられたテストケースのテスト結果を確認し、true ならば成功メッセージを表示し、false ならば失敗メッセージを表示する。

```
public static
   isOK : TestCase -> bool
   isOK (t) ==
     if GetTestResult (t)
     then FTestLogger`Success (t)
     else FTestLogger`Failure (t);
```

GetTestResult は、テスト結果を得る。

```
public static
   GetTestResult : TestCase -> bool
   GetTestResult (t) ==
     t.testResult;
```

GetTestName は、テスト名を得る。

```
public static
   GetTestName : TestCase -> seq of char
   GetTestName (t) ==
     t.testCaseName
```

```
end
```

FTestDriver

Test Suite : vdm.tc

Class : FTestDriver

Name	#Calls	Coverage
FTestDriver‘run	14	78%
FTestDriver‘isOK	81	70%
FTestDriver‘GetTestName	81	√
FTestDriver‘GetTestResult	81	√
Total Coverage		**78%**

2.31 FTestLogger

テストのログを管理する関数を提供する。

```
class
FTestLogger
values
  historyFileName = "VDMTESTLOG.TXT"
```

Success は、成功メッセージをファイルに追加し、標準出力に表示し、true を返す。

```
functions
public static
  Success : FTestDriver`TestCase -> bool
  Success (t) ==
    let message =
            FTestDriver`GetTestName (t)^"\tOK.\n",
        - = Fprint (message),
        - = Print (message) in
    true;
```

Failure は、失敗メッセージをファイルに追加し、標準出力に表示し、false を返す。

```
public static
  Failure : FTestDriver`TestCase -> bool
  Failure (t) ==
    let message = FTestDriver`GetTestName (t)^"\tNG.\n",
        - = Fprint (message),
        - = Print (message) in
    false;
```

SuccessAll は、全体成功メッセージをファイルに追加し、標準出力に表示し、true を返す。

```
public static
  SuccessAll : seq of char -> bool
  SuccessAll (m) ==
    let message = m^"\tOK!!\n",
        - = Fprint (message),
        - = Print (message) in
    true;
```

FailureAll は、全体失敗メッセージをファイルに追加し、標準出力に表示し、false を返す。

```
public static
```

```
  FailureAll : seq of char -> bool
  FailureAll(m) ==
    let message = m^"\tNG!!\n",
        - = Fprint(message),
        - = Print(message) in
    false;
```

Print は、標準出力に文字列を表示する。

```
public static
  Print : seq of char +> bool
  Print(s) ==
    new IO().echo(s);
```

Fprint は、現在ディレクトリの historyFileName で示されるファイルに文字列を表示する。

```
public static
  Fprint : seq of char +> bool
  Fprint(s) ==
    new IO().fecho(historyFileName,s,<append>)
```

Pr は、標準出力に文字列を表示するが、返値がない。

```
operations
public static
  Pr : seq of char ==> ()
  Pr(s) ==
    let - = new IO().echo(s) in
    skip;
```

Fpr は、現在ディレクトリの historyFileName で示されるファイルに文字列を表示するが、返値がない。

```
public static
  Fpr : seq of char ==> ()
  Fpr(s) ==
    let - = new IO().fecho(historyFileName,s,<append>) in
    skip
end
```

FTestLogger

Test Suite : vdm.tc

Class : FTestLogger

Name	#Calls	Coverage
FTestLogger`Pr	0	0%
FTestLogger`Fpr	0	0%
FTestLogger`Print	96	√
FTestLogger`Fprint	95	√
FTestLogger`Failure	0	0%

Name	#Calls	Coverage
FTestLogger‘Success	81	√
FTestLogger‘FailureAll	0	0%
FTestLogger‘SuccessAll	14	√
Total Coverage		**66%**

3 参考文献等

VDM++[2] は、1970 年代中頃に IBM ウィーン研究所で開発された VDM-SL[1] を拡張し、さらにオブジェクト指向拡張したオープンソース[*2]の形式仕様記述言語である。

参考文献

[1] CSK システムズ. VDM-SL 言語マニュアル. CSK システムズ, 第 1.1 版, 2007. Revised for VDMTools V7.1.

[2] CSK システムズ. VDM++ 言語マニュアル. CSK システムズ, 第 1.1 版, 2007. Revised for VDMTools V7.1.

[*2] 使用に際しては、（株）CSK システムズとの契約締結が必要になる。

索引