**GeoCalcVBの利用法**

実行形式を利用する場合

　\GeoCalcVB\bin\DebugフォルダにあるGeoCalcVB.exe（アプリケーション）ファイルをダブルクリックすれば，GeoCalcVBを起動させることができます．

Visual Basic環境がある場合

　GeoCalcVB.sln（Microsoft Visual Studio Solution）ファイルをダブルクリックすればVisual Basicが起動し，GeoCalcVBプロジェクトを開くことができます．

　プログラムはWindows Vista, 7, 8, 10で動作し，.NET Framework 4.0以降に対応しているVisual Studio 2010, 2012, 2013, 2015とそのExpressやCommunity Editionで編集できます．

**GeoCalcVBプログラムの解説**

　論文中ではプログラムの構造やプログラミング技法については詳しく述べることができませんでしたので，プログラムを公開する際にその詳細をプログラムに添えて解説する予定をしていました．ところが，プログラム開発者である大熊俊明氏が2016年9月22日に40歳の若さで急逝され，詳細な解説ができなくなってしまいました．

　基礎となる理論は論文に書かれているとおりですが，それを実現するプログラムは再帰アルゴリズムを活用してとても巧妙に作成されています．解読するためには何らかの指針が必要ですが，私（塩野）にはプログラムの解説は難しいです．幸い2015年9月10日時点での大熊氏による概要の説明文がありますので，それを以下に掲載します．詳細な解説とは言えませんが，プログラムを解読する上の参考にしてください．論文の論旨の変更に伴い，プログラムの改定（最終改訂：2016年5月19日）がありましたので，以下の解説がプログラムや論文と多少の不整合があるかもしれませんが，ご了承ください．なお，プログラムの実行時の振る舞いは論文に書かれている通りです．

塩野清治

**GeoCalcプログラム解説**

五大開発株式会社　大熊　俊明

最終更新 2015/08/17

目　次

[**全体構造 2**](#_Toc427587404)

[**Tクラスについて 2**](#_Toc427587405)

[**TSクラスについて 8**](#_Toc427587406)

[**ConsoleGeoCalcについて 14**](#_Toc427587407)

# 全体構造

GeoCalc.vbがライブラリ本体で，ConsoleGeoCalc.vbには，入力文字列を処理して，GeoCalc.vbを使って演算を行い，結果を表示するためのクラスがあります．

　GeoCalc.vbにはTとTSという2つのクラスがあります．Tはタプル1つを表し，TSは複数のタプルを表します．

例： (001,110,111)はTで保持可能

 (001,110,111)+(101, 110, 101)はTSで保持可能で1つ1つはT型となる．

ConsoleGeoCalc.vbは，ConsoleGeoCalcというクラスを持ち，このクラスのCalculateかCalculateAdvanceに計算したい文字列を渡すと計算結果が文字列で返ります．CalculateAdvanceの方が縮約計算まで行う点が異なります．

# Tクラスについて

次のメソッドを持ちます．一つずつ説明していきます．

 Private val As Long()

valがタプルの実際のlong型の値を保持する配列です．プログラム上は，クラスの末尾に置いてあります．（その理由は，プライベートな変数や関数は末尾に置く，という規約に従っているからです）

 ' 整数型の列でタプル内の値を指定する

 Public Sub New(ByVal ParamArray Values() As Long)

 ReDim val(Values.Length() - 1)

 Dim index As Long

 For index = 0 To Values.Length() - 1

 val(index) = Values(index)

 Next

 End Sub

 Valuesというので可変長の引数を受け取っています．

これは，コンストラクタです．すなわち，

 Dim t1 as T

 t1 = new T( array )

の形で使用します．arrayはdim array as long()というlong型の配列とします．

例えば，(001, 111, 110)を持つTを作るには，

 Dim t1\_array as long(2)

 t1\_array(0) = 1

 t1\_array(0) = 1+2+4

 t1\_array(0) = 2+4

 Dim t1 as T

 t1 = new T(t1\_array)

とします．

 ' タプルの指定位置の値を取り出す

 Public Function GetC(ByVal index As Long) As Long

 Return val(index)

 End Function

タプルの指定位置の値を取り出します．こういう値を表すのにCというポストフィクス文字を今後用います．

 ' タプル数を返す

 Public Function GetTupleSize() As Long

 Return val.Length()

 End Functio

　タプルの値の数を返す関数です．

例えば，(001, 110, 110)だったら値が3つあるので，3が返ります．

 ' 比較演算子を定義する

 Public Shared Operator =(ByVal t1 As T, ByVal t2 As T) As Boolean

 If t1.GetTupleSize <> t2.GetTupleSize Then

 Return False

 End If

 Dim index As Long

 For index = 0 To t1.GetTupleSize - 1

 If t1.GetC(index) <> t2.GetC(index) Then

 Return False

 End If

 Next

 Return True

 End Operator

2つのタプルが等しいかの判定を行うとき，=が使えるようにします．

例えば，

 Dim t1 as T, t2, as T

とあったときに，

 If ( t1 = t2 ) then

 XXXX　何らかの処理

 End If

と書けるようになります．

 Public Shared Operator <>(ByVal t1 As T, ByVal t2 As T) As Boolean

 Return Not (t1 = t2)

 End Operator

<> 演算子を定義しています．単純に=演算子の結果を反転しているだけです．

 ' 項同士のAnd演算の定義

 Public Function AndT(ByVal v1 As T) As T

 If GetTupleSize() <> v1.GetTupleSize Then

 Throw New ApplicationException("項の数の違うタプルのAndを計算しました")

 End If

 Dim ans As Long()

 ReDim ans(v1.GetTupleSize() - 1)

 Dim index As Long

 For index = 0 To v1.GetTupleSize() - 1

 ans(index) = v1.GetC(index) And GetC(index)

 Next

 Return New T(ans)

 End Function

2つのタプル間の共通集合を計算します．

 Dim t1 as T, t2 as T, t3 as T

があるとき，

 T3 = t1.AndT(t2)

というように使います．

 ' 複製を作成する

 Public Function Copy() As T

 Return New T(val)

 End Function

現在のタプルTの複製を作成します．valが値の配列なので，特に複雑なことはする必要がなくて，単にそれを用いた新たなTを作成して返すだけです．

 Dim t1 as T, t2 as T

 t1 = t2.Copy()

というように使います．

 ' すべて１のTを作成する

 Public Function One() As T

 Dim ret As Long()

 ReDim ret(val.Length - 1)

 Dim index As Long

 For index = 0 To val.Length - 1

 ret(index) = 7

 Next

 Return New T(ret)

 End Function

(111, 111, 111, …, 111)のようなタプルを作成します．

なお，タプルの値の数は，基準とするタプルと同じになります．

 Dim t1 as T, t2 as T

 t1 = t2.One()

というように使います．

 ' 特定の位置を反転する

 Public Function NegC(ByVal index As Long) As T

 Dim v1 As T

 v1 = One()

 v1.val(index) = val(index) Xor 7

 Return v1

 End Function

現在の位置を判定して，それ以外を111で埋めたタプルを作成します．補集合の計算用です．

例えば，(110, 001, 110)に対して1を指定すると，(111, 110, 111)が返ります．

 Dim t1 as T, t2 as T

 t1 = t2.NegC(1)

のように使います．

 '項のnot演算の定義

 Public Function NegT() As TS

 Dim ret As TS = New TS()

 Dim index As Long

 For index = 0 To GetTupleSize() - 1

 ret.AddSimple(NegC(index))

 Next

 Return ret

 End Function

補集合を計算します．上のプログラムに出てくるTSはあとで説明します（タプルの集合を扱えるクラスです）．プログラムがやっていることは，タプルの値の一つずつに補集合の計算をしながら，結果をTSに追加しているだけです．

 ' 文字列に変換する

 Public Overrides Function ToString() As String

 Dim ret As String

 ret = "("

 Dim index As Long

 For index = 0 To val.Length() - 1

 If index > 0 Then ret += ","

 ret += Convert.ToString(val(index) + 8, 2).Substring(1) ' "1???"という文字列直したあと，先頭の1を削除する

 Next

 Return ret + ")"

 End Function

　内部表現を可視化用の文字列に変換します．

 ' 和の公式が適用可能かをチェックする

 Public Function CanSimpleOr(ByVal t1 As T) As Boolean

 If t1.GetTupleSize() <> GetTupleSize() Then

 Return False

 End If

 Dim i As Long

 Dim cnt As Long

 cnt = 0

 For i = 0 To GetTupleSize() - 1

 If (val(i) <> t1.GetC(i)) Then

 cnt = cnt + 1

 If cnt >= 2 Then

 Return False

 End If

 End If

 Next

 Return True

 End Function

　和の公式が適用可能かをチェックします．具体的には，2つのタプルを比較して，値が異なる場所が2つ以上ある場合がダメ，と判定しています．

たとえば，(001, 110,101), (101, 101, 101)は2か所値が異なるので和の公式が適用不可です．

 ' 和の公式を適用する

 Public Function SimpleOr(ByVal t1 As T) As T

 Dim ans() As Long

 ReDim ans(GetTupleSize() - 1)

 Dim i As Long

 For i = 0 To GetTupleSize() - 1

 ans(i) = GetC(i) Or t1.GetC(i)

 Next

 Return New T(ans)

 End Function

　和の公式を適用します．単純に論理和をすべてのタプルの値同士に適用するだけです．

なお，前チェックとかは入ってないのでご使用のときにはご注意ください．

例えば，(001,010)と(001,101)ならば，(001, 111)になるわけです．

 ' ゼロかを判定する

 Public Function IsZero() As Boolean

 Dim i As Long

 For i = 0 To GetTupleSize() - 1

 If (val(i) = 0) Then

 Return True

 End If

 Next

 Return False

 End Function

　値が000を含むかをチェックしています．000を含んでいればTrueが返ります．

 ' Ωかを判定する

 Public Function IsOne() As Boolean

 Dim i As Long

 For i = 0 To GetTupleSize() - 1

 If (val(i) <> 7) Then

 Return False

 End If

 Next

 Return True

 End Function

　値がΩかを判定しています．タプルの値がすべて111ならばTrueとなります．

# TSクラスについて

　TSクラスはタプルの集合を表すクラスです．一つずつ説明していきます．

 Private val() As T

　プログラム上は，プライベートを末尾に書くという規約に従って末尾に書いてあります．

T型の変数の配列で，タプル集合を保持するのに使用します．

　 ' Tクラスの配列を指定する

 Public Sub New(ByVal ParamArray Values())

 ReDim val(Values.Length - 1)

 Dim index As Long

 For index = 0 To Values.Length - 1

 val(index) = Values(index)

 Next

 End Sub

　コンストラクタです．Valuesというのが可変長の引数になっていて，任意の数のTを受け取れます．すなわち，次のような感じで使用します．

 Dim t1 as T, t2 as T, t3 as T

 Dim t\_array as T(2)

 t\_array(0) = t1

 t\_array(1) = t2

 t\_array(2) = t2

 Dim ts1 as TS

 ts1 = new TS(t\_array)

tの配列を渡すことで，TSを構築することができます．

 ' コピーを作成する

 Public Function Copy() As TS

 Dim val\_copy() As T

 ReDim val\_copy(Size() - 1)

 Dim index As Long

 For index = 0 To Size() - 1

 val\_copy(index) = val(index).Copy()

 Next

 Return New TS(val\_copy)

 End Function

　自身の複製を作成します．なお，Tはオブジェクトなので，T自身のCopyを呼び出して複製を作成します（単純な代入だと同じオブジェクトになりますのでダメです）．

 Dim ts1 as TS, ts2 as TS

 ts1 = ts2.Copy()

 ' タプルの数を返す

 Public Function Size() As Long

 Return val.Length

 End Function

　タプル集合の要素数を返します．

例えば，

(001,110,111) + (010, 111, 101)

だったら2が返ります．

 ' 指定位置のTを返す

 Public Function GetT(ByVal index As Long) As T

 Return val(index)

 End Functio

　指定した位置のTが返ります．

 ts1 = (001,110,111) + (010, 111, 101)

に対して

 ts1.GetT(0)

をすると，

 (001,110,111)

が返ります．

 ' Andを計算する

 Public Function AndTS(ByVal t1 As T) As TS

 Dim ans As T()

 ReDim ans(Size() - 1)

 Dim index As Long

 For index = 0 To Size() - 1

 ans(index) = GetT(index).AndT(t1)

 Next

 Return New TS(ans)

 End Function

　タプル集合と1つのタプルの共通集合を計算して，タプル集合として返します．

 ' Andを計算する

 Public Function AndTS(ByVal ts1 As TS) As TS

 Dim index As Long

 Dim ans As TS = New TS()

 For index = 0 To ts1.Size - 1

 Dim ts\_tmp As TS

 ts\_tmp = AndTS(ts1.GetT(index)) ' 処理を任せている

 If index = 0 Then

 ans = ts\_tmp

 Else

 ans.AddSimple(ts\_tmp)

 End If

 Next

 Return ans

 End Function

2つのタプル集合の共通集合を計算して返します．

 ' Orを計算する

 Public Function OrTS(ByVal ts1 As TS) As TS

 Dim ret As TS

 ret = Copy()

 ret.AddSimple(ts1)

 Return ret

 End Function

　2つのタプル集合の和集合を計算します．特に処理はしていなくて，2つのタプル集合を単純に合体させるだけです．（縮約とかもなしです）

 ' 単純に要素を追加する．

 Public Sub AddSimple(ByVal t1 As T)

 ' サイズチェック

 If (val.Length > 0 AndAlso val(0).GetTupleSize <> t1.GetTupleSize()) Then

 Throw New ApplicationException("項の数の違うタプルを計算しました")

 End If

 ' 同一要素が存在しないかのチェック

 Dim index As Long

 For index = 0 To val.Length - 1

 If t1 = val(index) Then

 Return

 End If

 Next

 ' 追加

 ReDim Preserve val(val.Length)

 val(val.Length - 1) = t1

 End Sub

　同じ要素がすでにあるか，だけは，チェックしますが，それ以外は特になにもせずに，タプルをタプル集合に追加します．

 Dim ts1 as TS, t1 as T

 ts1.AddSimple(t1)

と書きます．ts1自身が変化します．

 ' 単純に要素を追加する．

 Public Sub AddSimple(ByVal ts1 As TS)

 Dim index As Long

 For index = 0 To ts1.Size - 1

 AddSimple(ts1.GetT(index))

 Next

 End Sub

　タプル集合をタプル集合に加えます．

 Dim ts1 as TS, ts2 as TS

 ts1.AddSimple(ts2)

と書きます．ts1自身が変化します．

 ' 文字列出力

 Public Overrides Function ToString() As String

 Dim str As String

 str = ""

 Dim index As Long

 For index = 0 To val.Length - 1

 If index > 0 Then

 str += "+" ' 最初以外に+を追加

 End If

 str += val(index).ToString

 Next

 Return str

 End Function

文字列としてタプル集合の内容を出力します．

 (001,101,111)+(001,110,110)

のような感じの文字列になります．すなわち，タプルを＋で接続した文字列になります．

 Public Sub DeleteT(ByVal index As Long)

 Dim j As Long

 For j = index To val.Length - 2

 val(j) = val(j + 1)

 Next j

 ReDim Preserve val(val.Length - 2)

 End Sub

　タプル集合の中で，指定したインデックス位置のタプルを削除します．

 ' 縮約計算

 Public Function Reduce() As TS

 Dim ans As TS

 ans = Copy()

 ' 和の公式

 Dim i As Long, j As Long

 Dim loopflag As Boolean

 loopflag = True

 While loopflag

 loopflag = False

 For i = 0 To ans.val.Length - 1

 If i >= ans.val.Length Then

 Exit For

 End If

 For j = i + 1 To ans.val.Length - 1

 If j >= ans.val.Length Then

 Exit For

 End If

 If ans.val(i).CanSimpleOr(ans.val(j)) Then

 ans.val(i) = ans.val(i).SimpleOr(ans.val(j))

 ans.DeleteT(j)

 loopflag = True

 End If

 Next

 Next

 End While

 ' 000化

 For i = 0 To ans.val.Length - 1

 If i >= ans.val.Length Then

 Exit For

 End If

 If ans.val(i).IsZero() And ans.val.Length > 1 Then

 ans.DeleteT(i)

 End If

 Next

 ' Φ化

 For i = 0 To ans.val.Length - 1

 If ans.val(i).IsOne() Then

 ReDim Preserve ans.val(0)

 ans.val(0) = ans.val(0).One()

 Exit For

 End If

 Next

 Return ans

 End Function

縮約計算を行います．

1. 和の公式の適用
2. 000の削除
3. 111かのチェック

の順番で行っています．

和の公式は1回適用したことで，新たな適用ができるようになる可能性もあるので，適用が一回も行われなくなるまで繰り返しています．

# ConsoleGeoCalcについて

文法は次のようになっています．いわゆる再帰下降構文解析法です．

{}は0以上の繰り返しを意味しています．”x”は例えばxそのものを表します．

expression = term { “+” term } # 和集合

 expression = term

 expression = term + term

 expression = term + ... + term

term = factor1 { “\*” factor1 } # 共通集合

 term = factor1

 term = factor1 \* factor1

 term = factor1 \* .... \* factor1

factor1 = factor2 | factor2 “’” # 補集合

 factor1 = factor2

 factor1 = factor2’

factor2 = “{“ expression “}” | tuple # {}の処理

 factor2 = tuple

 factor2 = {expression}

tuple = “(“ TV { “,” TV } “)” # タプル

 tuple = ( TV ,TV ,TV .... ,TV )

 { “,” TV }は　“,” TV　の繰り返し

　　 tuple = (TV) (001)など

 tuple = (TV,TV) (001,010)

 tuple = (TV,TV,TV) (010,100,110)

TV = V V V # 値

 TV=000,001,010,... Vの値が3個並んだもの

V = “0” | “1” # 値

　　 V=0 or V=1

例

 expression = term + .... + term

 expression = factor1 \* factor1 + factor1

 expression = factor2 \* factor2’+ factor2

 expression = tuple \* tuple’+ {tuple’+ tuple’}

 expression = {(110,101)\*(010,011)’}+{(001,110)+(110,101)}’

　上の規則をほぼそのままの形でプログラムに直すだけで，上の構文解析ができてしまいます．

プログラム中の関数名とは，

V →　Parse1 0,1

TV　→　Parse2 101

tuple → Parse3 (101,110,001)

factor2 → Parse4 {(101,110,001)+(011,111,100)’}

factor1 → Parse5 {(101,110,001)+(011,111,100)’}’

term → Parse6 　　　　(101,111,110)\*{(101,110,001)+(011,111,100)’}’

expression → Parse7 　　　 (001,101,111)+{(101,110,001)+(011,111,100)’}’

と対応しています．

例えば，expressionに相当するParse7は，下記のようになります．

 ' +の項の処理

 Public Function Parse7() As GeoCalc.TS

 Dim ts1 As GeoCalc.TS, ts2 As GeoCalc.TS

 ts1 = Parse6()

 While True

 PassSpace()

 If Not IsEndInput() AndAlso \_input(\_index) = "+" Then

 ' +の処理をする

 \_index = \_index + 1

 ts2 = Parse6()

 ts1 = ts1.OrTS(ts2)

 Else

 Exit While

 End If

 End While

 Return ts1

 End Function

 expression = term { “+” term }

をそのままプログラムに直したものであることが読んでいただければわかると思います．

ほかの場所も同様です．

以上