Artificial Intelligence and TupleSpace of ultranetwork

Masaaki Yamaguchi

クラウドにデータベースを構築しておいて、この構築した多様体を数式で表現したコード通りのデータが、この多様体において、作用素関数として実行されるとする。この多様体を実現したデータが表現されている環境自体を表せられるソースとして、TupleSpace が辞書を書き換えることができないことを利便して、どのデータも上書きされないことによって、前後の記憶が無駄なコードが作用されないことを表現できる。

作用素環プログラミングとして、半静性型宣言子をつくる。この宣言子は、スクリプト型プログラミング言語では、この型を作り上げた時点で、その宣言した環境としての多様体がデータベースの仕様として、宣言した以後のソースコードがこのコード自体の性質を反映させることが多様体を表現した後の、配列、ハッシュ、文字列、ポインタ、ファイル構造体、オブジェクト、数値、関数、正規表現、行列、統計、微分、積分、この微分、積分は関数とは別の文字列と数値処理として、行列と統計をこの表現としての多様体として、微分、積分を数列を応用とした極限値としてソースコードをコンピュータにおいて、実行、表現、存在できるコンピュータ上だけにとどまらないプログラミング言語として、調べられる。この作用素としての半静性型宣言子は、スクリプト言語において、重要な研究として、動的と静的な宣言子として、なぜ静的宣言子が動的スクリプトで必要とされているかが、Streem と Ruby を学んでいく段階で浮かび上がった課題として、私は Ruby をオブジェクト指向を学んだ結果が、この作用素環プログラミングをプログラム思考でコンピュータに人工知能を生成出来て、人体の量子コンピュータを模擬出来て、その上に、FPGA までも実行できるアスペクト型人工知能スクリプト言語が、この多様体を数式を文字列としてだけではなく、電気信号としての表現体としてコンピュータ上に実現できることを研究課題として、生まれている。

Omega::DATABASE を tuplespace としてスクリプトに書き上げているソースをデータベースの下地とする。これをコンピュータに多様体として表現、実行、流れとして、動的に実行する。この実行した後に、スクリプト言語の動作を停止した場合は、ガベージコレクションとして破棄されるとする。この動作している状態のときに、同時に実行される関数、オブジェクト、文字出力は、このときに同時に起動している多様体の性質をウェブのネットワーク上で多様体の記述されている規則、ルールに則ってプログラミング言語でコンピュータに作用させている、最終的な産物のゼータ関数としてのガンマ関数の大域的微分多様体を熱エントロピー値として、この熱値の性質として分類、整列される TupleSpace 上の関数の群論として、なにがコンピュータ上だけでなく、存在論だけにとどまらない電気信号かが、数学と情報科学で研究されるべきと、この多様体を調べることが必要と、目下の課題になっている。

現実の世界として、この世界を架空化する空間が同型としてのフェルミオンとボソンが、この空想上での入れ物に電気信号としての文字列がバーチャルネットワークに出力されて、この出力される文字列と電気信号が架空の性質として、物体や生命に現実の世界としての相対的な実存を特徴、成分、性質、分類としてコンピュータに文字列として命を吹き込む機能をプログラミング言語で生成されたバーチャルコードによって生み出せる可能性を秘めている。

```
Omega::DATABASE[tuplespace]
{
      Z \supset C \bigoplus \nabla R^{+}, \nabla(R^{+}
      \cap E^{+}) \ni x, \Delta(C \subset R) \ni x
      M^{+}_{-}\bigoplus R^{+}, E^{+} \in
      \bigoplus \nabla R^{+}, S^{+}_{-} \subset R^{+}_{2},
      V^{+}_{-} \times R^{+}_{-} \cong {V \over S}
      C^{+} \cup V^{+}_{-} \ni M_{1}\bigoplus \nabla C^{+}_{-},
      Q \supseteqq R^{+}_{-},
      Q \subset \bigoplus M^{+}_{-},
   \bigotimes Q \subset \zeta(x), \bigoplus \nabla C^{+}_{-} \cong M_3
     R \subset M_3,
   C^{+} \bigoplus M_n, E^{+} \cap R^{+},
   E_2 \bigoplus E_1, R^{-} \subset C^{+}, M^{+}_{-}
     C^{+}_{-}, M^{+}_{-}\nabla C^{+}_{-}, C^{+}\nabla H_m,
  E^{+} \nabla R^{+}_{-}, E_2 \nabla E_1,
   R^{-} \nabla C^{+}_{-}

      [- \Delta v + \nabla_{i} \nabla_{j} v_{ij} - R_{ij} v_{ij}
  - v_{ij} \nabla_{i} \nabla_{j} + 2 < \nabla f, \nabla h>
  + (R + \nabla f^2)({v \over 2} - h)]

      S^3, H^1 \times E^1, E^1, S^1 \times E^1, S^2 \times E^1,
      H^1 \times S^1, H^1, S^2 \times E
}
```

クラウドにおけるデータベースを多様体が機能する仕組みからデータの相互関係と各データの処理対応が数学における多様体からソースコード化できる。

まず始めに、ソースコードを記述する人が定義したデータベースをライブラリーとして、動的にスクリプト言語に取り込む

```
import Omega::Tuplespace < DATABASE
{
  {\bigoplus M^{+}_{-} -> =: \nabla R^{+} \nabla C^{+}}-< [construct_emerge_equation.built]
  >> VIRTUALMACHINE[tuplespace]
  => {regexpt.pattern |w|
      w.scan(equal.value) [ > [\nabla \int \int \nabla_{i}\nabla_{j} f \circ g(x)]]
       equal.value.shift => tuplespace.value
       w.emerged >> |value| value.equation_create
       w <- value
       w.pop => tuplespace.value
     }
}
```

多様体の式をバーチャルマシンに方程式としてと、データベースとして

```
{\bigoplus M^{+}_{-} -> =: \nabla R^{+} \nabla C^{+}}-< [construct_emerge_equation.built]
```

```
>> VIRTUALMACHINE[tuplespace]
```

多様体の式を分岐したリストで、配列に生成される方程式の再構築とバーチャルマシンに >> で入力する。

```
=> {regexpt.pattern |w|
    w.scan(equal.value) [ > [\nabla \int \int \nabla_{i}\nabla_{j} f \circ g(x)]]
     equal.value.shift => tuplespace.value
     w.emerged >> |value| value.equation_create
     w <- value
     w.pop => tuplespace.value
   }

   }
```

このバーチャルマシンに入力されたデータを正規表現で共通要素を抽出して、
これも配列に入っている定義されている多様体へ、数値解析として>と入力する。
この共通データをデータの端から取り除く値を tuplespace **の値としてリスト化する。**
この抽出されてデータを、データベースに取り込んでいる多様体の規則から
トリガーとして機能を発動させる。この多様体の値を再び正規表現として<-と
入力する。このデータベースの全データを取り入れた段階で再構築して、
生成し直す。

もとのデータ >> 対象物のデータ、>> は文字入力機能を表す。
もとのデータ >- 対象物のデータ、>- はデータの分岐の流れを作る。

```
{\vee (\int \nabla_{i}\nabla_{j} (R + \Delta f)^2)
  \over \exists (R + \Delta f)} -> =: variable array[]
>> VIRTUAL_MACHINE[tuplespace]
=> {regexpt.pattern |w|
    w.emerged => tuplespace[array]
    w <- value
    w.pop => tuplespace.value
   }
```

多様体を入力する配列を -> =: 変数 array[] **と表す。**
>> は、データベースに配列として入力する。
このデータベースに機能しているデータを正規表現として扱うように{equation}=>
{regexpt.pattern}**へデータを流す。そのあとに、自動でデータを更新する。**

```
Omega.DATABASE[tuplespace]->w.emerged >> |value| value.equation_create
{
  w.process <- Omega.space
  {=>
      cognitive_system :=> tuplespace[process.excluded].reload
      assembly_process <- w.file.reload.process
```

```
      => : [regexpt.pattern(file)=>text_included.w.process]
  }
}
```

データベースから正規表現で生成された変数値から、それにポインタされた
方程式を、データベースをもとで生成する。この生成された中で、
ソースコードを正規表現にプロセス、マルチスレッド化して、外部のデータを
後ろからポインタとして、連結する。`w.process <- Omega.space` として
上の表現として表している。
これもデータベースとして扱い、そのコード実行の中で、作用素環機能として、
だが、機能ではなく、機能をポインタで指されているアドレスに存在定義されている
`cognitive_system` を機能を実現させる一種の合言葉として、
`tuplespace[process.excluded]` へデータを`:=>` を使い、流す。これを `reload` する。
`assembly_process` も変数のような定数で表せられて、この変数に `w.file.reload.process`
としてポインタを当てる。この当てられた `assembly_process` を配列のデータベースへ
正規表現をファイルに記述されているデータとして再取り込みを行う。

```
Omega.DATABASE[tuplespace]->w.emerged >> |list| list.equation_create
{
  w.process <- Omega.space
  {=>
    poly w.process.cognitive_system :=> tuplespace[process.excluded].reload
    homology w.process :=> tuplespace[process.excluded].reload
    mesh.volume_manifold :=> tuplespace[process.excluded].reload
    \nabla_{i}\nabla_{j} w.process.excluded :=> tuplespace[process.excluded].reload
    {\exp[\int \int (R + \Delta f)^2 e^{-x \log x}dV}.emerge_equation.reality{|repository|
     repository.regexpt.pattern => tuplespace[process.excluded].reload
     tuplespace[process.excluded].rebuild >> Omega.DATABASE[tuplespace]
    {\imaginary.equation => e^{\cos \theta + i\sin \theta}} <=> Omega.DATABASE[tuplespace]
    {{d \over df}F ==> {d \over df}{1 \over {(x \log x)^2 \circ (y \log y)
    ^{1 \over 2}}}dm}.cognitive_system.reload
    :=> [repository.scan(regexpt.pattern) { <=> btree.scan |array| <-> ultranetwork.attachment}
    repository.saved
    }
  }
}
```

データベースから連想リスト構造の方程式を生成して、
このデータの `tuple` から、外部でのスペースに記述されているデータをポインタとして指して、
取り込む。
この取り込んでいるデータを、作用素環の半静性宣言子としての、`poly,homolgy,equation` が記述さ
れているソース
の式を使って、データを各ポインタを指しているデータ自体にリンクとして双対性をプログラミングし
ていく。
`:=>, >> ==> ,<=> .emerge_equation.reality, .reload, .cognitive_system, .reload, .saved`
の各ポインタを指すための代入子、入力子、等号入力子、倒置入力子、　記述されている方程式を生成
する、
それを実行する。再取り込み、連想配列生成、保存を各レシーバはオブジェクトから保持している機能
を呼び起こせられる。

```
import ultra_database.included
def < this.class::Omega.DATABASE[first,second,third.fourth] end
 def.first.iterator => array.emerge_equation
 def.second.iterator => array.emerge_equation
 def.third.iterator => array.emerge_equation
 def.fourth.iterator => array.emerge_equation
 _ struct_ {
            Omega.iterator => repository.reload
 }
end
   typedef _ struct_ :Omega.aspective
end
```

今までのデータベースをウルトラネットワークとかして、取り込む。この多様体がデータベースとして宣言されている情報空間へ関数のメソッドとして、データベースに記述されている機能としてのメソッドとして各正規表現を配列に入っている文字列から方程式として、多様体のデータベースの単体量へポインタを介して、関数のメソッドのハッシュを作り、このポインタへの各要素のデータベースをリポジトリとしての構造体へとアスペクト指向として、関数定義する。

```
Omega::DATABASE[reload]
{
  [category.repository <-> w.process] <=> catastrophe.category.selected[list]
  list.distributed => ultra_database.exist ->
  w.summurate_pattern[Omega.Database]
  btree.exclude -> this.klass
  list.scan(regexpt.pattern) <-> btree.included
  list.exclude -> [Omega.Database]
  all_of_equation.emerged <=> Omega.Database
  {
    list.summuate -> Omega.Database.excluded
  }
}
```

今までのデータをデータベースにリロードして、その中で、不変性を見つけて分類していく。
この分類された連想配列によるリスト構造をウルトラネットワークへ双対性 => をつかって、
-> と統合されるべきパターンへと流す。これを btree 構造体にポインタをつなげて、リスト化して、
各リストを再びデータベースへとつなげる。
今までの方程式をデータベースの中の多様体に入れて、相互に比較してリスト構造体を再編成する。

```
list.distributed => {
      {\bigoplus \nabla M^{+}_{-}}.constructed <-> Omega.Database[import]
      {=>
         each_selected :file.excluded
```

```
        }
    }
```

この再編成されたリストを自分が導いた方程式が、どの範疇のデータで、何の方程式かを、多様体から意思が生成された認知でもある場の理論として、判断させて、未知の理論を多様体からの人工知能で見つける。

これらのデータベース化されたリストから、レシーバでもある、前からの宣言と後ろからの、レシーバで
オブジェクトとして、リスト化したデータへ、以下の式たちを入力させる。`equation_manifolds.scan(value) |value|`
と=`|value|` **以下で数式たちを代入**=**している。**

```
Omega::DATABASE[tuplespace] >> list.cognitive_system |value|
= { x^{{1 \over 2} + iy} = [f(x) \circ g(x), \bar{h}(x)]/ \partial f\partial g\partial h
 x^{{1 \over 2} + iy} = \mathrm{exp}[\int \nabla_{i}\nabla_{j}f(g(x))g'(x)/
 \partial f\partial g]

 \mathcal{O}(x) = \{[f(x)\circ g(x) , \bar{h}(x)], g^{-1}(x)\}

   \exists [\nabla_{i} \nabla_{j} (R + \Delta f), g(x)] = \bigoplus_{k=0}^{\infty}
\nabla \int \nabla_{i} \nabla_{j}f(x)dm

   \vee (\nabla_{i} \nabla_{j} f) = \bigotimes \nabla E^{+}

    g(x,y) = \mathcal{O}(x)[f(x) + \bar{h}(x)] + T^2 d^2 \phi

  \mathcal{O}(x) = \left( \int [g(x)] e^{-f}dV \right)^{'} - \sum \delta (x)
   \mathcal{O}(x) = [\nabla_{i}\nabla_{j}f(x)]^{'} \cong {}_{n}C_{r} f(x)^{n}
   f(y)^{n-r} \delta (x,y),
   V(\tau) = \int [f(x)]dm/ \partial f_{xy}

   \square \psi = 8 \pi G T^{\mu\nu}, (\square \psi)^{'} = \nabla_{i}\nabla_{j}
   (\delta (x) \circ G(x))^{\mu\nu}
 \left({p \over c^3} \circ {V \over S}\right), x^{{1 \over 2} + iy} = e^{x \log x}

   \delta (x) \phi = {\vee [\nabla_{i}\nabla_{j} f \circ g(x)] \over
   \exists (R + \Delta f)}



 {}_{-n}C_{r} = {}_{{1 \over i}H\psi} C_{\hbar \psi} + {}_{[H, \psi]} C_{-n - r}
   {}_{n}C_{r} = {}_{n}C_{n-r}



 \int \int {1 \over (x \log x)^2}dx_m \to \mathcal{O}(x) =
 [\nabla_{i}\nabla_{j}f]'/\partial f_{xy}


 \bigcup_{x=0}^{\infty} f(x) = \nabla_{i}\nabla_{j}f(x) \oplus \sum f(x)
```

```
 = \bigoplus \nabla f(x)
 \nabla_{i}\nabla_{j} f \cong \partial x \partial y \int
 \nabla_{i}\nabla_{j} f dm
     \cong \int [f(x)]dm
  \cong \{[f(x),g(x)],g^{-1}(x)\}
 \cong \square \psi
 \cong \nabla \psi^2
 \cong f(x \circ y) \le f(x) \circ g(x)
 \cong |f(x)| + |g(x)|


  \delta (x) \psi = <f,g>\circ |h^{-1}(x)|
  \partial f_x \cdot \delta (x) \psi = x
  x \in \mathcal{O} (x)
  \mathcal {O} (x) = \{[f \circ g, h^{-1}(x)], g(x) \}

   \lim_{n \to \infty} \sum_{k=n}^{\infty} \nabla f = [\nabla \int
 \nabla_{i}\nabla_{j} f(x) dx_m, g^{-1}(x)] \to \bigoplus_{k=0}^{\infty}
 \nabla E^{+}_{-}
  = M_{3}
  = \bigoplus_{k=0}^{\infty} E^{+}_{-}
  dx^2 = [g^2_{\mu\nu},dx], g^{-1} = dx \int \delta(x)f(x)dx
  f(x) = \mathrm{exp}[\nabla_{i}\nabla_{j}f(x),g^{-1}(x)]
  \pi(\chi,x) = [i\pi (\chi,x), f(x)]
  \left({g(x) \over f(x)}\right)^{'} =
  \lim_{n \to \infty} {g(x) \over f(x)}
           = {g'(x) \over f'(x)}


   \nabla F = f \cdot {1 \over 4}|r|^2
  \nabla_{i}\nabla_{j} f = {d \over dx_i}
{d \over dx_j}f(x)g(x)
 D^2 \psi = \nabla \int (\nabla_{i}\nabla_{j} f)^2 d\eta
 E = m c^2, E = {1 \over 2}mv^2 - {1 \over 2}kx^2, G^{\mu\nu} =
 {1 \over 2}\Lambda g_{ij},
\square = {1 \over 2}kT^2

 \mathrm{ker} f / \mathrm{im} f \cong S^{\mu\nu}_m,
 S^{\mu\nu}_m = \pi (\chi,x) \otimes h_{\mu\nu}

 D^2 \psi = \mathcal{O} (x)\left({p \over c^3} +
 {V \over S}\right), V(x) = D^2\psi \otimes M^{+}_3

 S^{\mu\nu}_{m} \otimes S^{\mu\nu}_{n} =
 - {2R_{ij} \over V(\tau)}[D^2\psi]

 \nabla_{i}\nabla_{j}[S^{mn}_1 \otimes S^{mn}_2] =
 \int {V(\tau) \over f(x)}[D^2 \psi]
  \nabla_{i}\nabla_{j}[S^{mn}_1 \otimes S^{mn}_2] =
  \int {V(\tau) \over f(x)}\mathcal{O}(x)

 z(x) = {g(cx + d) \over f(ax + b)}h(ex + l)
   = \int{V(\tau) \over f(x)}\mathcal{O}(x)
```

```
 {V(x) \over f(x)} = m(x), \mathcal{O}(x) = m(x)[D^2\psi(x)]
 {d \over df}F = m(x), \int F dx_m = \sum_{k=0}^{\infty} m(x)

 \mathcal{O}(x) = \left( [\nabla_{i}\nabla_{j}f(x)]\right)^{'}
  \cong {}_{n}C_{r}(x)^{n}(y)^{n-r} \delta(x,y)
 (\square \psi)' = \nabla_{i}\nabla_{j}(\delta(x) \circ
 G(x))^{\mu\nu} \left({p \over c^3} \circ
{V \over S} \right)
 F^m_t = {1 \over 4}g^{2}_{ij}, x^{{1 \over 2} + iy} = e^{x \log x}
 S^{\mu\nu}_m \otimes S^{\mu\nu}_n = G_{\mu\nu} \times T^{\mu\nu}

  S^{\mu\nu}_m \otimes S^{\mu\nu}_n  = -{2 R_{ij} \over V(\tau)}[D^2 \psi]

 S^{\mu\nu}_m  = \pi(\chi,x) \otimes h_{\mu\nu}

 \pi (\chi,x) = \int \mathrm{exp}[L(p,q)]d\psi
 ds^2 = e^{-2\pi T|\phi|}[\eta + \bar{h}_{\mu\nu}]dx^{\mu\nu}dx^{\mu\nu} +
 T^2 d^2\psi

    M_3 \bigotimes_{k=0}^{\infty} E^{+}_{-} = \mathrm{rot}
    (\mathrm{div} E, E_1)
    = m(x), {P^{2n} \over M_3} = H_3(M_1)

 \exists [R + |\nabla f|^2]^{{1 \over 2} + iy}
 = \int \mathrm{exp}[L(p,q)]d\psi
 = \exists [R + |\nabla f|^2]^{{1 \over 2} + iy} \otimes
 \int \mathrm{exp}[L(p,q)]d\psi +
N\mathrm{mod}(e^{x \log x})
 = \mathcal{O}(\psi)

 {d \over dt}g_{ij}(t) = - 2 R_{ij}, {P^{2n} \over M_3}
 = H_3(M_1), H_3(M_1) = \pi (\chi, x) \otimes h_{\mu\nu}
 S^{\mu\nu}_{m} \times S^{\mu\nu}_{n}
 = [D^2\psi] , S^{\mu\nu}_{m} \times S^{\mu\nu}_{n}
 = \mathrm{ker}f/\mathrm{im}f,  S^{\mu\nu}_{m} \otimes
 S^{\mu\nu}_{n} = m(x)[D^2\psi], {-{2R_{ij} \over V(\tau)}} = f^{-1}xf(x)
 f_z = \int \left[ \sqrt{\begin{pmatrix} x & y & z \\
   u & v & w \end{pmatrix} \circ
   \begin{pmatrix} x & y & z \\
   u & v & w \end{pmatrix}}_{}\right]dxdydz,
   \to f_z^{1 \over 2} \to (0,1)\cdot(0,1) = -1,i =
 \sqrt{-1}
{\begin{pmatrix} x,y,z
    \end{pmatrix}}^2 = (x,y,z)\cdot(x,y,z) \to - 1




 \mathcal{O}(x) = \nabla_{i}\nabla_{j} \int e^{{2 \over m}\sin \theta
 \cos \theta} \times {N \mathrm{mod}
(e^{x \log x})
```

```
\over \mathrm{O}(x)(x + \Delta |f|^2)^{1 \over 2}}
 x \Gamma(x) = 2 \int |\sin 2\theta|^2d\theta,
 \mathcal{O}(x) = m(x)[D^2\psi]

 \lim_{\theta \to 0}{1 \over \theta} \begin{pmatrix} \sin \theta \\
   \cos \theta \end{pmatrix}
   \begin{pmatrix} \theta & 1 \\
   1 & \theta \end{pmatrix}
   \begin{pmatrix} \cos \theta \\
   \sin \theta \end{pmatrix}
   =  \begin{pmatrix} 1 & 0 \\
   0 & - 1 \end{pmatrix},
f^{-1}(x) x f(x) = I^{'}_m, I^{'}_m = [1,0] \times [0,1]


 i^2 = (0,1) \cdot (0,1),|a||b|\cos \theta = -1,
 E = \mathrm{div}(E,E_1)
 \left({\{f,g\} \over [f,g]}\right)^{'} = i^2,  E = mc^2, I^{'} = i^2

 \mathcal{O}(x) = || \nabla \int [\nabla_{i}\nabla_{j} f
 \circ g(x)]^{{1 \over 2} + iy}|| , \partial r^n
||\nabla||^2 \to \nabla_{i}\nabla_{j} ||\vec{v}||^2

 \nabla^2 \phi

 \nabla^2 \phi = 8 \pi G \left({p \over c^3} + {V \over S}\right)

 (\log x^{1 \over 2})^{'} = {1 \over 2}{1 \over (x \log x)},
(\sin \theta)^{'} = \cos \theta, (f_z)^{'} = i e^{i x \log x},
{d \over df}F = m(x)

 {d \over df}\int \int{1 \over (x \log x)^2}dx_m
 + {d \over df}\int \int {1 \over (y \log y)^{1\over2}}dy_m
= {d \over df} \int \int \left({1 \over (x \log x)^2}
+ {1 \over (y \log y)^{1 \over 2}}\right)dm
 \ge {d \over df}\int \int \left({1 \over
 (x \log x)^2 \circ (y \log y)^{1 \over 2}}\right)dm
 \ge 2h

 {d \over df}\int \int \left({1 \over (x \log x)^2 \circ
 (y \log y)^{1 \over 2}}\right)dm  \ge \hbar
 y = x, xy = x^2, (\square \psi)^{'} = 8 \pi G
 \left({p \over c^3}\circ{V \over S}\right)
 \square \psi = \int \int \mathrm{exp}[8 \pi G(\bar{h}_{\mu\nu}
 \circ \eta_{\mu\nu})^{\mu\nu}]dmd\psi,
 \sum a_k x^k = {d \over df}\sum \sum {1 \over a^2_k f^k}dx_k
 \sum a_k f^k = {d \over df}\sum \sum
{\zeta(s) \over a_k}dx_{km},
 a^2_kf^{1 \over 2}\to \lim_{k \to 1}a_k f^k = \alpha

   ds^2 = [g_{\mu\nu}^2, dx]
   M_2
   ds^2 = g_{\mu\nu}^{-1}(g^2_{\mu\nu}(x) - dx g_{\mu\nu}^2)
```

```
  M_2
   = h(x) \otimes g_{\mu\nu}d^2x - h(x) \otimes dx g_{\mu\nu}(x),
  h(x) = (f^2(\vec{x}) - \vec{E}^{+})
   G_{\mu\nu} = R_{\mu\nu}T^{\mu\nu},
   \partial M_2 = \bigoplus \nabla C^{+}_{-}
    G_{\mu\nu}   equal   R_{\mu\nu}  {d \over dt}g_{ij} = - 2 R_{ij}
r = 2 f^{1 \over 2}(x)
     E^{+} = f^{-1}xf(x),
   h(x) \otimes g(\vec{x}) \cong {V \over S},
  {R \over M_2} = E^{+} - {\phi}
     = M_3 \supset R,
   M^{+}_2 = E^{+}_{1} \cup E^{+}_{2} \to E^{+}_1 \bigoplus E^{+}_2
     = M_1 \bigoplus \nabla C^{+}_{-}, (E^{+}_{1} \bigoplus E^{+}_{2})
     \cdot (R^{-} \subset C^{+})
     {R \over M_2} = E^{+} - \{\phi\}
     = M_3 \supset R
     M^{+}_3 \cong h(x) \cdot R^{+}_3
  = \bigoplus \nabla C^{+}_{-},
  R = E^{+} \bigoplus M_2 - (E^{+} \cap M_2)
     E^{+} = g_{\mu\nu}dxg_{\mu\nu},
   M_2 = g_{\mu\nu}d^2x,
   F = \rho g l \to {V \over S}
     \mathcal{O}(x) = \delta(x)[f(x) + g(\bar{x})] + \rho g l,
   F = {1 \over 2}mv^2 - {1 \over 2}kx^2,
   M_2 = P^{2n}
      r = 2f^{1 \over 2}(x),
  f(x) = {1 \over 4}\|r\|^2

     V = R^{+}\sum K_m, W = C^{+}\sum^{\infty}_{k=0} K_{n+2},
     V/W = R^{+}\sum K_m / C^{+}\sum K_{n+2}
     = R^{+}/C^{+} \sum{x^k \over a_k f^k(x)}
     = M^+_{-}, {d \over df} F = m(x), \to M^{+}_{-}, \sum^{\infty}_{k=0}
     {x^k \over a_k f^k(x)} = {a_k x^k \over
  \zeta(x)}

     {\{f,g\} \over [f,g]} = {fg + gf \over fg - gf},
  \nabla f = 2, \partial H_3 = 2, {1 + f \over 1 - f} = 1,
  {d \over df} F = \bigoplus \nabla C^{+}_{-}, \vec{F} =
  {1 \over 2}
     H_1 \cong H_3 = M_3

   H_3 \cong H_1 \to \pi(\chi,x), H_n, H_m =
   \mathrm{rank}(m,n), \mathrm{mesh}(\mathrm{rank}(m,n)) \lim \mathrm{mesh} \to 0
   (fg)' = fg' + gf', ({f \over g})' = {{f'g - g'f} \over g^2},
   {\{f,g\} \over [f,g]} = {(fg)' \otimes dx_{fg} \over
({f \over g})' \otimes g^{-2}dx_{fg}}
   = {{(fg)'\otimes dx_{fg}} \over ({f \over g})' \otimes g^{-2}dx_{fg}}
   = {d \over df} F


     \hbar\psi = {1 \over i} H \Psi, i[H,\psi] = - H \Psi, {\{f,g\} \over [f,g]} = (i)^2

     [\nabla_{i} \nabla_{j} f(x), \delta(x)] = \nabla_{i} \nabla_{j}
```

```
   \int f(x,y)dm_{xy}, f(x,y) = [f(x), h(x)] \times [g(x), h^{-1}(x)]
   \delta(x) = {1 \over f'(x)}, [H, \psi] = \Delta f(x),
   \mathcal{O}(x) = \nabla_{i} \nabla_{j} \int \delta(x)f(x)dx
   \mathcal{O}(x) = \int \delta(x)f(x)dx

   R^{+} \cap E^{+}_{-} \ni x, M \times R^{+} \ni M_3, Q \supset C^{+}_{-},
   Z \in Q \nabla f, f \cong \bigoplus_{k=0}^{n} \nabla C^{+}_{-}
   \bigoplus_{k=0}^{\infty} \nabla C^{+}_{-} = M_1, \bigoplus_{k=0}^{\infty}
   \nabla M^{+}_{-} \cong E^{+}_{-},
 M_3 \cong M_1 \bigoplus_{k=0}^{\infty} \nabla {V^{+}_{-} \over S}
   {P^{2n} \over M_2} \cong \bigoplus_{k=0}^{\infty}
   \nabla C^{+}_{-}, E^{+}_{-} \times R^{+}_{-} \cong M_2
   \zeta(x) = P^{2n} \times \sum_{k=0}^{\infty} a_k x^k,
   M_2 \cong P^{2n}/\mathrm{ker}f, \to \bigoplus \nabla C^{+}_{-}
   S^{+}_{-} \times V^{+}_{-} \cong {V \over S} \bigoplus_{k=0}^{\infty}
   \nabla C^{+}_{-}, V^{+} \cong M^{+}_{-} \bigotimes S^{+}_{-},
   Q \times M_1 \subset \bigoplus \nabla C^{+}_{-}
   \sum_{k=0}^{\infty} Z \otimes Q^{+}_{-} = \bigotimes_{k=0}^{\infty} \nabla M_1
   = \bigotimes_{k=0}^{\infty} \nabla C^{+}_{-} \times
   \sum_{k=0}^{\infty} M_1, x \in R^{+} \times C^{+}_{-}
 \supset M_1, M_1 \subset M_2 \subset M_3

S^3, H^1 \times E^1, E^1, S^1 \times E^1, S^2 \times E^1, H^1 \times S^1,
 H^1, S^2 \times E$.
\bigoplus \nabla C^{+}_{-} \cong M_3, R \supset Q, R \cap Q,
R \subset M_3, C^{+} \bigoplus M_n, E^{+} \cap R^{+}$
 M^{+}_{-}\nabla C^{+}_{-}, C^{+}\nabla H_m, E^{+} \nabla R^{+}_{-}, E_2 \nabla E_1 $
 $ R^{-} \nabla C^{+}_{-} $.     {\nabla \over \Delta} \int x f(x) dx,
 {\nabla R \over \Delta f}, \square = 2{\int {(R + \nabla_{i} \nabla_{j} f)^2
 \over -(R + \Delta f)}}e^{-f}dV
   \square = {\nabla R \over \Delta f}, {d \over dt}g_{ij}
   = \square \to {\nabla f \over \Delta x}, (R +
 |\nabla f|^2)dm \to -2(R + \nabla_{i} \nabla_{j} f)^2 e^{-f}dV
   x^n + y^n = z^n \to \nabla \psi^2 = 8 \pi G T^{\mu\nu},
   f(x + y) \ge f(x) \circ f(y)
   \mathrm{im}f / \mathrm{ker}f = \partial f, \mathrm{ker}f
   = \partial f, \mathrm{ker}f / \mathrm{im}f \cong
 \partial f, \mathrm{ker}f = f^{-1}(x)xf(x)
   f^{-1}(x)xf(x) = \int \partial f(x) d(\mathrm{kerf}) \to \nabla f = 2
   _{n}C_{r} = {}_{n}C_{n-r}  \to \mathrm{im}f / \mathrm{ker}f
   \cong \mathrm{ker}f / \mathrm{im}f



  $ \sum^{\infty}_{k=0}a_k f^k = T^2d^2 \phi $. this equation $ a_k \cong
  \sum^{\infty}_{r=0} {}_n C_r $.
   V/W = R/C \sum^{\infty}_{k=0}{x^k \over a_k f^k}, W/V = C/R
   \sum^{\infty}_{k=0}{a_k f^k \over x^k}
   V/W \cong W/V \cong R/C(\sum^{\infty}_{r=0} {}_nC_{r})^{-1}
   \sum^{\infty}_{k=0} x^k
 This equation is diffrential equation, then $ \sum^{\infty}_{k=0} a_k f^k $
 is included with $ a_k \cong \sum^{\infty}_{r=0} {}_nC_{r} $
   W/V = xF(x), \chi(x) = (-1)^k a_k, \Gamma(x) = \int e^{-x} x^{1 -t}dx,
```

```
      \sum^{n}_{k=0}a_k f^k = (f^k)'
      \sum^{n}_{k=0}a_k f^k = \sum^{\infty}_{k~0} {}_{n}C_{r} f^k
      = (f^k)',
    \sum^{\infty}_{k=0} a_k f^k = [f(x)],
    \sum^{\infty}_{k=0} a_k f^k = \alpha, \sum^{\infty}_{k=0}
    {1 \over a_k f^k}, \sum^{\infty}_{k=0} (a_k f^k)^{-1} = {1 \over 1 - z}

       {\int \int {1 \over (x \log x)(y \log y)}dxy} =
       {{{}_nC_{r} xy} \over {({}_nC_{n-r}
    (x \log x)(y \log y))^{-1}}}
       = ({}_nC_{n-r})^2 \sum_{k= 0}^{\infty}({1 \over x \log x}
       - {1 \over y \log y})d{1 \over nxy} \times {xy}
       = \sum_{k=0}^{\infty} a_k f^k
       = \alpha
}
```

これまでのデータベース化された機能のもとでもある方程式たちを構造体として、
まとめて、=> [tuplespace] **としてポインタを当てる。**

```
 _ struct_ :asperal equation.emerged => [tuplespace]
tuplespace.cognitive_system => development -> Omega.Database[import]
value.equation_emerged.exclude >- Omega.Database[tuplespace]
```

この連想ポインタは tuplespace **自体をオブジェクト化して、レシーバの**.cognitive_system **から**
実装段階で、Omega **データベース化する。**
このデータベースの仮の方程式、未定義な式をデータベースから多様体の仕組みを利用して、
データベースから分岐して、value **のオブジェクトとしてポインタを当てる。**

以下のソースコードは、今まで扱っている多様体のデータを使って、アプリケーションプログラミングとして、即席スクリプト言語を DSL として書いている。

```
Omega::DataBase <-> virtual_connect(VIRTUALMACHINE)
{
  blidge_base.network => localmachine.attachment
  :=> {
       dhcp.etc_load_file(this.klass) {|list|
        list.connect[XWin.display _ <- xhost.in(regexpt.pattern)]
        {
          ultranetwork.def _struct {
           asperal_language :this.network_address.included[type.system_pattern]
            {|regexpt.pattern|
              <- w.scan
```

```
                    |each_string| <= { ipv4.file :file.port
                                       subnetmask :file.address
                                                      file.port <=> file.address
                                       FILE *pointer
                                       int,char,str :emerge.exclude > array[]
                                       BTE.each_string <-> regexpt.pattern
                                        {
                                          development => file.to_excluded
                                            file.scan => regexpt.pattern
                                              this.iterator <-> each_string
                                               file.reloded => [asperal_language.rebuild]
                                        }
                                     }
          }
        }
      }
     }
    }
}
```

```
class Ultranetwork
 def virtual_connect
  load :file => {
   asperal :virtual_machine.attachment
   {
     system.require :file.attachment
     <- |list.file| :=> {
         tk.mainloop <- [XWin -multiwindow]
         startx => file.load.environment
           in { [blidge_base | host_base].connect(wmware.dhcp)
                net_work.connect.used[wireshark.demand => exclude(file)]
          }
     }
   }
  }
 end
```

def < method **として、メソッドを** def **へリファクタリング機能をつかって、**
def **へと以下の** method **たちは取り込まれる。**
これの作用は、 def **が** one class **並の等号シングレトンとして機能する。**

```
 def < blidge_base.network.connect
   {
    dhcp.start => {
                   host_name <-> localhost_name {|list|
                     list.exist(connect_type)
                      {
                        <- : tty :xhost -display => list.exist
```

```
                              [virtual_connect].list->host :terminal
                       }
                    }
   }
 end


 def < host_base.ethernet.connect
   {

                   host_name.connect => local_network
                  }
   }
 end




 def < etc.load_file
   {
    etc.include(inetd.rc)
     {
       virtual_connect(VIRTUAL_MACHINE){|list|
        list.attachment(etc.load_file)
       }
     }
   }
 end

 mainloop{
  def.virtual_connect => xhost.localmachine
  {
   xhost.client <-> xhost.server
  }
  def.network.type <- [Omega.DATABASE] end
  def.etc.load_file.attachment(VIRTUAL_MACHINE) end
 end
end


class UltraNetwork::DATABASE import OMEGA.TUPLESPACE
  def load_file >- VIRTUAL_MACHINE
   { in . => attachment_device |for|
   for.load -> acceptance.hardware
   virtual_machine.new
    {
    tk.loop-> start
    XWin -multiwindow
    if dwm <-> new_xwin.start
    localhost :xhost :display -x
    xdisplay :-> [preset :XFree.demand>=needed
    for.set_up
    install_process >- tar -xvfz "#{load_file}" <-> install_attachment
```

```
  ]
  else if
  only :new_xwin.start
  localhost :xhost :multiwindow . { in
  display -x
  attachment :localhost -client
  from -client into
  server.XWin -attachment}
  end
  condition :{ in .=>
  check->[xdisplay.install_process]}
end

def < network_rout
        wireshark.start -> ethernet.device >- define rout
              rout.ipstate do |file|
                 file.type <- encoding XWin -filesystem
                 file.included >- make kernel_system.rebuild
                 file.vmware.start do |rout|
                 rout.blidgebase | rout.hostbase
         -> file.install
            file.address_ipstate
            => {"{file}" :=> dwm.state_presense
            virtual_machine.included[file]
    }
end

def < launcher_application
       network_rout.new
       |file|
       file.attachment => { in .
       new_xwin.start :=> file.included
       demand.file <- success_exit}
end

def < terminal_port
       network_rout.new
       launcher_application.new |rout|
       rout.acceptance {
       vmware.state.process |new_rout|
       new_rout : attachment.class <-> dwm.state_attachment
       new_rout -> condition.start_wmware.process}
end


def < kterm_port
        launcher_application.new
        def.included[DATABASE]
        |rout|
        rout.attachment <- |new_rout|
        new_rout.attachment do
        install.condition < rout.def.terminal_port.exclude[file]
end
```

```
  main_loop :file do
             kterm_port.excluded :=> VIRTUAL_MACHINE
             |new_rout| start do
             rout.process -> network_rout.rout [
             file,launcher_application, terminal_port, kterm_port].def < included
             |file|
             file.all_attachment: file_type :=> encoding-utf8
  end
end



class < def {
      pholograph_data[] = [R,V,S,E,U,M_n,Z_n,Q,C,N,f,g]
      source_array <- pholograph_data[]
}

def > operator_data[] = {nabla,nabla_i nabla_j,Delta,partial,
                         d, int, cap,cup,ni,in,chi,oplus,otimes,bigoplus,bigotimes,d /over df,
end

def > manifold_emerge

         c = def.inject >- source_array times def.operator_data[]
 repository_data <=> c{

 c.scan(/tupplespace[]/)
 import |list| list{
    kerf = -2 \int (R + nabla_i nabla_j f)^2e^{-f}dV
    kerf / imf
    =< {d \over df}F}
     }
         equals_data =~ /list/
             list.match(/"#{c}"/) {|list|
             list.delete
             jisyo_data_mathmatics <=> list{
            list.emerge => {asperal function >- pholograph_data[] times repository_data
                   =< list.update}
            }
                   ln -s operator_named <= {list}
                    define _struct |list|
                          -> list.element -> manifold_emerge
                          =>  list.reconstruct > def.inject /^"#{pattern}"/}
end


import Omega::Tuplespace < Database
{
  {\bigoplus \nabla M^{+}_{-}}.equation_create -> asperal :variable[array]
   :=> [cognitive_system <-> def < VIRTUALMACHINE.terminal
                                   {
```

```
                                      [ipv4.bloadcast.address :
                                        ipv4.network.adress].subnetmask
                                       <-> file.port.transport_import :
                                               Omega[tuplespace]
                                    }
}

_struct _ Omega[tuplespace] >> VIRTUALMACHINE.terminal.value


class < def.VIRTUALMACHINE.system_environment



             file.reload[hardware] => file.exclude >> file.attachment
             {=>
                |file|
                  file.port(wireshark.rout <-> {file.port.transport_export
                  :=> Omega[tuplespace]}
                        assembly_process.file.included >- file.reloaded
                             :- |file.environment| {=>
                                            file.type? :=> exist
                                              file.regexpt.pattern[scan.flex]
                                                   => |pattern|
                                                         <->
                                                           file.[scan.compiler]
                                 }
                         end
                 end
               file <<
              }
}

Omega::Database[tuplespace]
{
  cognitive_system |: -> { DATABASE.create.regexpt_pattern >-
     cognitive_system[tuplespace].recreated >- : =< DATABASE.value
      >> system_require.application.reloaded[tuplespace]
         } : _struct _ def.VIRTUALMACHINE.terminal >> {
             ||machine.attachment|| <-> OBJECT.shift => system.reloaded
             . in {
                     : _struct _ class.import :-> require mechanics.DATABASE
                        {|regexpt_pattern| :|-> aspective _union _
                         def _union _}
                  }
             }

   end

}
```

```
system.require <- import library.DATABASE
{
  Omega[tuplespace]
  {
       cognitive_system : VIRTUALMACHINE.equality_realized
       {|regexpt_pattern| => value | key [ > cognitive_system.loop.stdout]
            value : display -bash :xhost -number XWin.terminal
            key   : registry.edit :=> {[cognitive_system.reloaded]}
       }
  }
}


_union _ => DATABASE[tuplespace].aspective_reloaded
_union _ :fx | -> |regexpt_pattern| => {
                     VIRTUALMACHIE.recreated-> _union _ |
                     _struct _ def.DATABASE.recreated <- fx
                  >> DATABASE[tuplespace].rebuild
}

DATABASE[tuplespace] -< {[ > aimed.compiler | aimed.interpreter] | btree.def.distributed >-
                         aimed[tuplespace]}
aimed[tuplespace] -< btree.class.hyperrout_ struct _ => Omega::Database[tuplespace].value
  sheap_ union _ :aspective | -> Omega[tuplespace]: | aimed[tuplespace].differented_review
}

aimed[tuplespace].process => DATABASE[tuplespace].reloaded
aimed.different | aimed.stdout >> vale | key [ > cognitive_system.loop.stdin] {|pattern|
                             pattern.scan(value : aimed[def.value]
                                key   : aimed[def.key])
                } _ struct _ : flex | interpreter.system
                   => expression.iterator[def.first,def.second,def.third,def.fourth]
                      { def < Omega[tuplespace]
                        def.cognitive_system |: -> DATABASE[tuplespace] | aimed[tuplespace]
}

Omega::Tuplespace < DATABASE
{=>
   norm[Fx] -> . in for def.all_included < aimed[tuplespace].each_scan([regexpt_pattern]
   <->
                   DATABASE[tuplespace]) << streem database.excluded
   >- more_pattern.scan(value : aimed[def.value]

   key :aimed[def.key])
               . in { _struct _ :flex | interpreter.system
                   => expression.iterator[def.all.each -> |value, key|
                                  included >- norm[Fx]|[DATABASE[tuplespace]
  ,aimed[tupespace]] |
                                   finality : aimed[tuplespace], DATABASE[tuplespace]

   : -> def.included(in_all)
```

```
                                    {
                                        def.key | def,value => [DATABASE].recompile
       & make install
                                     : in_all -> _struct _ :aspective :tuplespace
    : all_homology_created}
                     }
}



def < Omega::Tuplespace[DATABASE]
 def.iterator -> |klass,define_method,constant,variable,infinity_data : -> finite_data|
         def.each_klass?{|value, key|
            _struct _ :aspective -> tuplespace :all_homology_recreated :make menuconfig
            {=+
               def.key -> aimed[def.key],def.value -> aimed[def.value] {|list|
                   list.developed => <key,value> | <aimed[$`,$']
                    -> _union _ :value,key : _struct _
                    <- (_union _ <-> _struct _ +)
               begin
                  def.key <-> aimed[value]
                  case :one_ exist :other :bug
                  {
                     result <-> def.key
                     {
                       differented :DATABASE[tuplespace]
                     }
                     return :tuplespace.value.shift -> included<tuplespace>
                  else if
                  :other :bug
                  {
                    success_exit <- bug[value]
                    {
                      cognitive_system.scan(bug[value])
                      {
                       {[e^{-f}[{2 \int (R + \nablaf^2) \over -(R + \Delta f)}e^{-f}dV}
       .created_field
                          {=>
                              regexpt.pattern \native_function <-> euler-equation
                               {
                                  $variable =< diff e^{-f} >- $'
                                    all_included <- def.key <-> aimed[value]
                                    $variable - all_included.diff
                                \summuate_manifold.recreated
       <- \native_function : euler-equation
                               }
                          }
                       }
                    } _union _ :cognitive_system.rebuild(one_ exist)
                 }
                }
                ensure
```

```
                {
                    return :success_exit
                    => Tuplespace[DATABASE]
                }
               }
             }
         end
 end
}
```

```
 int
streem_style {
  :Endire <- [ADD,EVEN,MOD,DEL,MIX,INCLUDED,EXCLUDED,EBN,EXN,EOR,EXOR,
             SUM,INT,DIFF,PARTIAL,ROUND,HOMOLOGY,MESH]
  Endire.interator -> {def < :Endire.element, -> def.means_each{x -> expression.define.included
  def.each{x -> case :x.each => :lex.include_ . in [ > [x.all_expire] ]}

}

main_loop {
  FILE *fp :=> streem_style.address_objective_space
  fp.each{x -> domain_specific_language_style_included[array]}
  array << streem.DATABASE[tuplespace]
  array.each{[tuplespace] -> aimed[tuplespace] | OMEGA_DATABASE[tuplespace]}.excluded <-> array
  def.key <-> def.value => {x -> stdin | stdout |=> streem_style <- def.each.klass.value}
}
```

リバイザを使うと、独自のタプルスペースで一時保存の書き換えの分派スクリプトが出来て、
その応用で、例外処理機能として、そのスクリプトを使うと、独自に機能拡張できる。
書き換え機能自体、書き換えたいとおもっている好きなところを書き換えられる。
構文解析器も文字抽出器も全部書き換えられる。
その書き換えたのをライブラリとして、データベースに取り込むと、この部分が例外処理機能の
おかげで、タプルスペースが働き、今まで使ってきた機能と一線を画しする。

```
@reviser : def < OmegaDatabase[tuplespace].mechanism
{
  aspective : _union _ {
       int streem_style : [ > [def.each{x -> stdin | stdout > display :xhost in XWin -multiwind
       {
         Endire <- [ADD,EVEN,ODE,EXOR,XOR,DEL,DIFF,PARTIAL,INT].included > struct _ :-> _union
         Endire.each{def.value -> def.key :hash.define}.included > _union}
        }
}

@reviser : def.reconstructed.each{_union <-> _struct _.recreated} : [def.del - def.before_deter
```

```
import perl.lib | python.lib <-> ruby.lib
{
int @reviser : def.each{x -> x.klass |-> $variable in $stdin | $stdout}
.developed >= {
                          ping localhost -> blidgebase <-> hostbase.virtualmachine.attachment
                          {
                              xhost :display -> streem_style.value
                              networkconnect.hostbase -> localarea.virtualmachine
                          } :connected -> networkrout : flow_to :localhost.attachment
  }
 }
```

```
_struct : def < hostbase.virtualmachine.attachment => : networkrout.area.build

@reviser <-> def.add [ < _struct]
@reviser : def.each{listmenu -> listlink | unlinklist > [developed -> {def.key , def.value}.cur
@reviser <-> def.rebuild [ < _struct]

@reviser.def.<value|key>networkrout-> def.present
def.present.flow_to -> hostbase.rout << networkrout.data.<value|key>

XWin -multiwindow <-> networkrout.data[$‘,$’]
def < $’
@reviser <-> def.present.state
@reviser.def.each{x | -> key.rebuild | value.rebuild}.flow_to :redefined
```

```
def < OmegaDatabase[tuplespace]
{
  FILE *fp -> cmd.value : cmd.key {fp |-> syncronized.file[tuplespace] | aimed.file[tuplespace]
  cmd.key => [ > fp.($‘:$’)] <-> registry.excluded<fp.file[cmd.state]>
}

def.each{fp|-> def.first,def.second,def.third,def.fourth}

cmd _struct : {
[ ^C-O : ^C-X-F, exit.cmd : ^C-X-C, shift-up : ^C-P, shift-down : ^C-N]
}

cmd _union : def.restructed
keyhook.cmd <- : [_struct ]
```

```
{
  @reviser :def._struct <-> def. _union

}
```