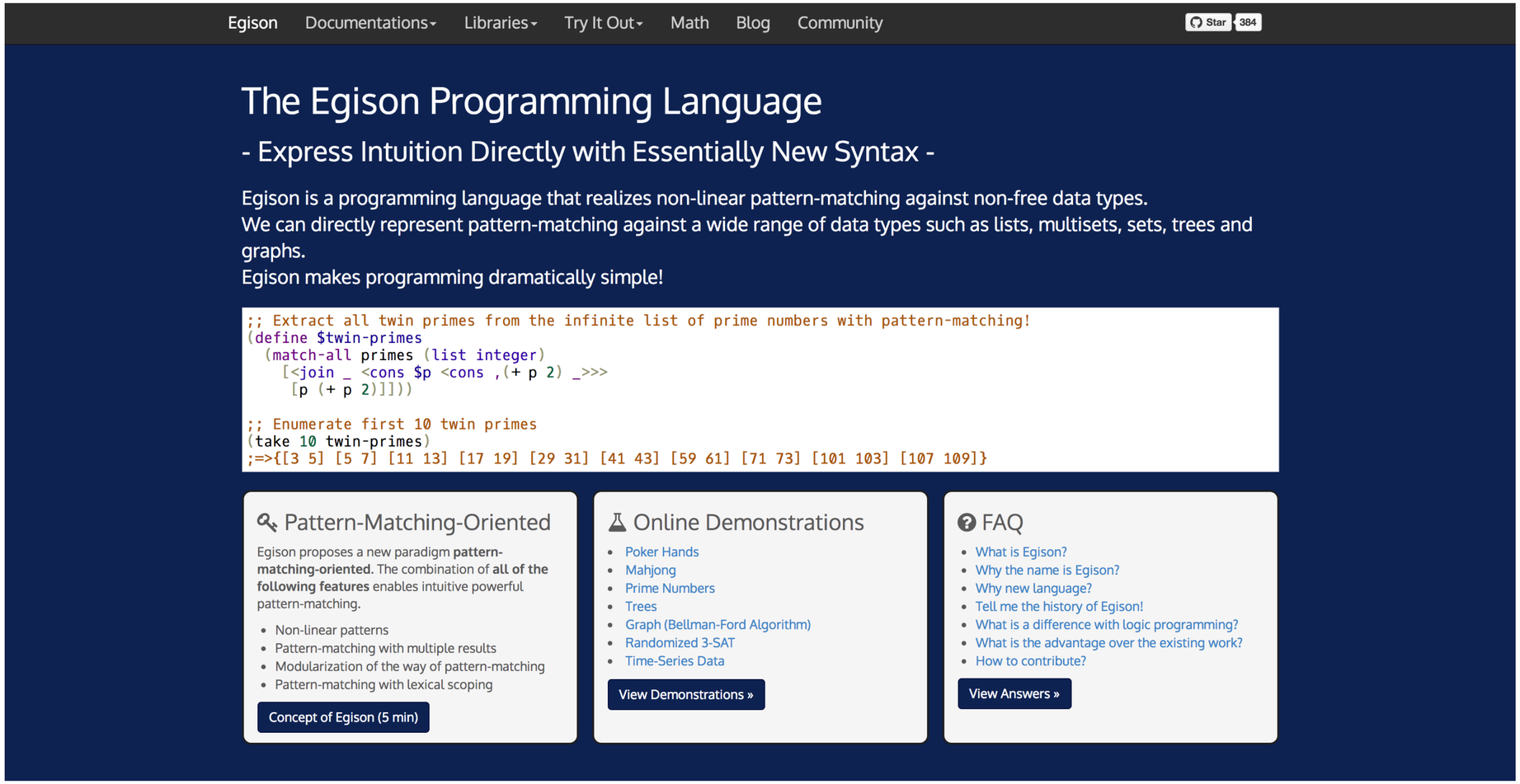
Scalar and Tensor Parameters for Importing Notations of Differential Geometry into Programming

v1 Nov.19.2017

Satoshi Egi
Rakuten Institute of Technology
Rakuten, Inc.

Rakuten

The Egison Programming Language



We are refining Egison paper to propagate the theory of Egison all over the world! Please check the latest copy of the paper on arXiv.org.

https://www.egison.org

Table of Contents

Table of Contents

Aim of Egison

My aim is to create a language that can represent directly all algorithms that can be discovered.

Currently, my biggest challenge is to improve Egison in order to represent directly calculations that appear in mathematical physics.

Features of Egison

Pattern-matching against the wider range of data types.

Customizable symbolic computation using Egison pattern-matching.

Tensor index notation in programming.

Features of Egison

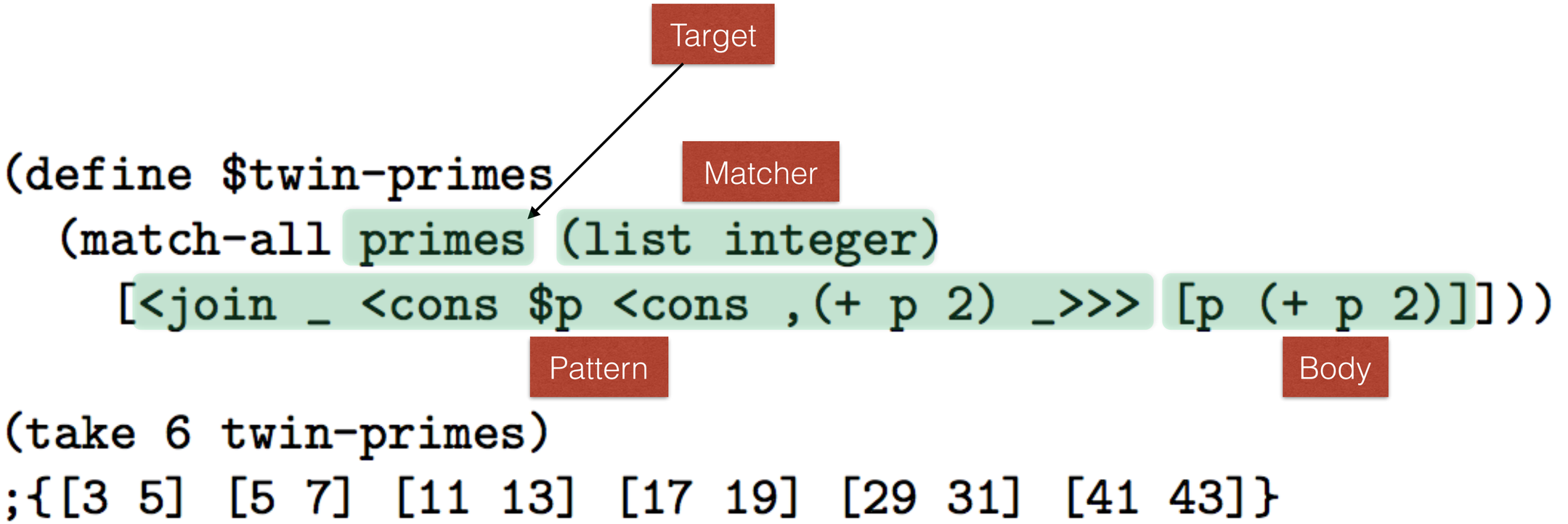
Pattern-matching against the wider range of data types.

Customizable symbolic computation using Egison pattern-matching.

Tensor index notation in programming.

Twin Primes

$$\langle \textit{match-all-expr} \rangle ::= \text{`(match-all'} \ \langle \textit{target} \rangle \ \langle \textit{matcher} \rangle \ \langle \textit{match-clause} \rangle \ \text{`)'}$$



Poker Hands

```
(define $poker-hands
  (lambda [$cs]
    (match cs (multiset card)
      {[<cons <card $s $n>
         <cons <card ,s ,(- n 1)>
          <cons <card ,s ,(- n 2)>
           <cons <card ,s ,(- n 3)>
            <cons <card ,s ,(- n 4)>
             <nil>>>>>>
        <Straight-Flush>]
       [<cons <card _ $n>
         <cons <card _ ,n>
          <cons <card _ ,n>
           <cons <card _ ,n>
            <cons _
             <nil>>>>>>
        <Four-of-Kind>]
       [<cons <card _ $m>
         <cons <card _ ,m>
          <cons <card _ ,m>
           <cons <card _ $n>
            <cons <card _ ,n>
             <nil>>>>>>
        <Full-House>]
       [<cons <card $s _>
         <cons <card ,s _>
          <cons <card ,s _>
           <cons <card ,s _>
            <cons <card ,s _>
             <nil>>>>>>
        <Flush>]
       [<cons <card _ $n>
         <cons <card _ ,(- n 1)>
          <cons <card _ ,(- n 2)>
           <cons <card _ ,(- n 3)>
            <cons <card _ ,(- n 4)>
             <nil>>>>>>
        <Straight>]
       [<cons <card _ $n>
         <cons <card _ ,n>
          <cons <card _ ,n>
           <cons _
            <cons _
             <nil>>>>>>
        <Three-of-Kind>]
       [<cons <card _ $m>
         <cons <card _ ,m>
          <cons <card _ $n>
           <cons <card _ ,n>
            <cons _
             <nil>>>>>>
        <Two-Pair>]
       [<cons <card _ $n>
         <cons <card _ ,n>
          <cons _
           <cons _
            <cons _
             <nil>>>>>>
        <One-Pair>]
       [<cons _
         <cons _
          <cons _
           <cons _
            <cons _
             <nil>>>>>>
        <Nothing>]}))))
```

are patterns.

Poker Hands - Straight Flush

```
(define $poker-hands
  (lambda [$cs]
    (match cs (multiset card)
      {[<cons <card $s $n>
        <cons <card ,s ,(- n 1)>
         <cons <card ,s ,(- n 2)>
          <cons <card ,s ,(- n 3)>
           <cons <card ,s ,(- n 4)>
            <nil>>>>>>
       <Straight-Flush>]
       [<cons <card _ $n>
        <cons <card _ ,n>
```

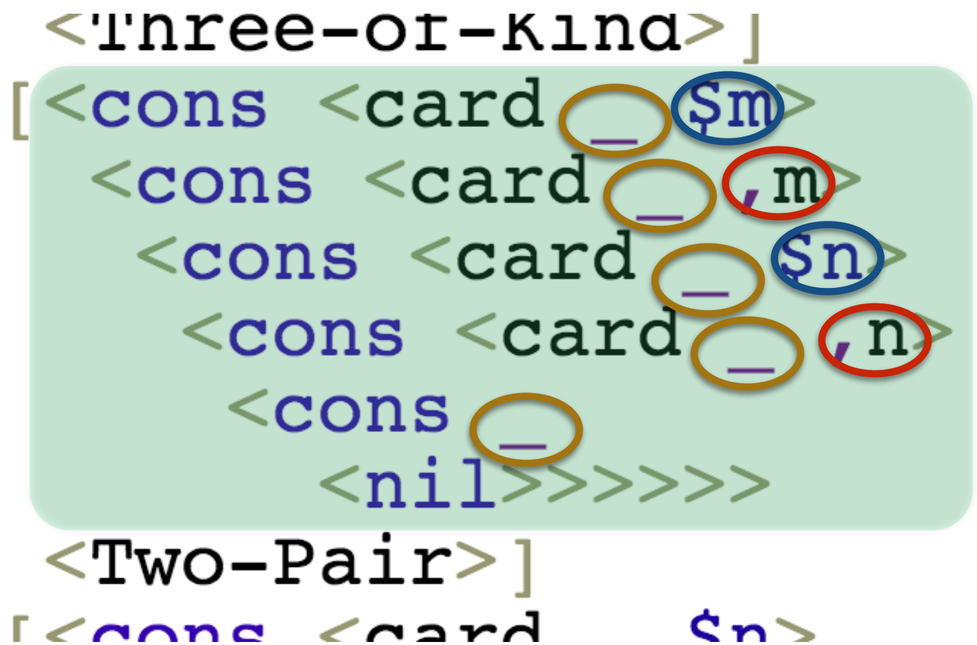
Poker Hands - Straight Flush

```
(define $poker-hands
  (lambda [$cs]
    (match cs (multiset card)
      {[<cons <card $s $n>
        <cons <card ,s ,(- n 1)>
         <cons <card ,s ,(- n 2)>
          <cons <card ,s ,(- n 3)>
           <cons <card ,s ,(- n 4)>
            <nil>>>>>>
       <Straight-Flush>]
       [<cons <card _ $n>
         <cons <card _ ,n>
```

Poker Hands - Two Pair

```
<Three-of-Kind>]
[<cons <card _ $m>
 <cons <card _ ,m>
  <cons <card _ $n>
   <cons <card _ ,n>
    <cons _
     <nil>>>>>>
 <Two-Pair>]
[<cons <card _ $n>
```

Poker Hands - Two Pair



Multiset Matcher

```
(define $multiset
  (lambda [$a]
    (matcher
      {[<nil> []
        {[{} {[]}]
         [_ {}]}]
       [<cons $ $> [a (multiset a)]
        {[$tgt (match-all tgt (list a)
                  [<join $hs <cons $x $ts>> [x {@hs @ts}]])]}]
       [,$val []
        {[$tgt (match [val tgt] [(list a) (multiset a)]
                  {[[<nil> <nil>] {[]}]
                   [[<cons $x $xs> <cons ,x ,xs>] {[]}]
                   [[_ _] {}]})]}]
       [$ [something]
        {[$tgt {tgt}]}]
       })))
```

Multiset Matcher - Cons Pattern

```
(define $multiset
  (lambda [$a]
    (matcher
      {[<nil> []
        {[{} {[]}]
         [_ {}]}]
       [<cons $ $> [a (multiset a)]
        {[$tgt (match-all tgt (list a)
                 [<join $hs <cons $x $ts>> [x {@hs @ts}]])]}]
       [,$val []
        {[$tgt (match [val tgt] [(list a) (multiset a)]
                 {[[<nil> <nil>] {[]}]
                  [[<cons $x $xs> <cons ,x ,xs>] {[]}]
                  [[_ _] {}]})]}]
       [$ [something]
        {[$tgt {tgt}]}]
       })))
```

Multiset Matcher - Cons Pattern

```
(define $multiset
  (lambda [$a]
    (matcher
      {[<nil> []
        {[{} {[]}]
         [_ {}]}]
       [<cons $ $> [a (multiset a)]
        {[$tgt (match-all tgt (list a)
                 [<join $hs <cons $x $ts>> [x {@hs @ts}]])]}]
       [,$val []
        {[$tgt (match [val tgt] [(list a) (multiset a)]
                 {[[<nil> <nil>] {[]}]
                  [[<cons $x $xs> <cons ,x ,xs>] {[]}]
                  [[_ _] {}]})]}]
       [$ [something]
        {[$tgt {tgt}]}]
       })))
```

Next patterns	Next matchers	Next targets
<cons $ $>	[a (multiset a)]	{[1 {2 3}] [2 {1 3}] [3 {1 2}]}

Meaning of Patterns for Collections before Egison

nil pattern: Match with an empty collection.

cons pattern: Divide a collection into **the head element** and the rest.

List	Multiset	Set
`{[1 {2 3}]}`	Applicable only to Lists.	

join pattern: Divide a collection into **the head part** and the rest.

List	Multiset	Set

`{[{} {1 2}]`
`[{1} {2}]`
`[{1 2} {}]}`
	Applicable only to Lists.	

Meaning of Patterns for Collections Generalized by Egison

nil pattern: Match with an empty collection.

cons pattern: Divide a collection into **an element** and the rest.

List	Multiset	Set

`{[1 {2 3}]}`	`{[1 {2 3}]`
	` [2 {1 3}]`
	` [3 {1 2}]}`
		`{[1 {1 2 3}]`
		` [2 {1 2 3}]`
		` [3 {1 2 3}]}`

join pattern: Divide a collection into **a part** and the rest.

List	Multiset	Set

`{[{} {1 2}]`
` [{1} {2}]`
` [{1 2} {}]}`
	`{[{} {1 2}]`
	` [{1} {2}]`
	` [{2} {1}]`
	` [{1 2} {}]}`
		`{[{} {1 2}]`
		` [{1} {1 2}]`
		` [{2} {1 2}]`
		` [{1 2} {1 2}]}`

Rakuten

Features of Egison

Pattern-matching against the wider range of data types.

Customizable symbolic computation using Egison pattern-matching.

Tensor index notation in programming.

Symbolic Computation

```
(* (+ x y) (+ x y))
;(+ x^2 (* 2 x y) y^2)

(** (+ x y) 3)
;(+ x^3 (* 3 x^2 y) (* 3 x y^2) y^3)

(** (+ x y) 4)
;(+ x^4 (* 4 x^3 y) (* 6 x^2 y^2) (* 4 x y^3) y^4)

(** (+ 1 i) 4)
;-4

(sqrt 4)
;2

(* (sqrt 2) (sqrt 3))
;(sqrt 6)

(* (sqrt 2) (sqrt 6))
;(* 2 (sqrt 3))
```

Simplification - cos(θ)^2 + sin(θ)^2 = 1, ω + ω^2 = -1, …

```
(+ (cos θ)^2 (sin θ)^2)
;1

(+ w w^2)
;-1

(+ (rtu 5) (rtu 5)^2 (rtu 5)^3 (rtu 5)^4)
;-1
```

7th Roots of Unity

$$\cos\left(\frac{2\pi}{7}\right) = \frac{1}{6}\left(-1 + \sqrt[3]{\frac{7+21\sqrt{3}i}{2}} + \sqrt[3]{\frac{7-21\sqrt{3}i}{2}}\right)$$

```
(define $z (rtu 7))
(define $a11 (+ z^1 z^6))
(define $a12 (+ z^2 z^5))
(define $a13 (+ z^3 z^4))
(define $b10 (+ a11 a12 a13))
(define $b11 (+ a11 (* w a12) (* w^2 a13)))
(define $b12 (+ a13 (* w a11) (* w^2 a12)));(* w b11)
(define $b13 (+ a12 (* w a13) (* w^2 a11)));(* w^2 b11)
(define $b14 (+ a11 (* w a13) (* w^2 a12)))
(define $b15 (+ a12 (* w a11) (* w^2 a13)));(* w b14)
(define $b16 (+ a13 (* w a12) (* w^2 a11)));(* w^2 b14)
(define $b10' b10)
(define $b11' (rt 3 (* b11 b12 b13)))
(define $b14' (rt 3 (* b14 b15 b16)))
(define $a11' (/ (+ b10' b11' b14') 3))
(define $z1' (fst (q-f' 1 (* -1 a11') 1)))
z1'
;(/ (+ -1 (rt 3 (+ 14 (* 21 w))) (rt 3 (+ -7 (* -21 w)))
;      (sqrt (+ -35
;               (* -2 (rt 3 (+ 14 (* 21 w))))
;               (* -2 (rt 3 (+ -7 (* -21 w))))
;               (rt 3 (+ 14 (* 21 w)))^2
;               (rt 3 (+ -7 (* -21 w)))^2
;               (* 2 (rt 3 (+ 14 (* 21 w))) (rt 3 (+ -7 (* -21 w))))))))
;   6)
```

Differential Operator

```
(define $∂/∂
  (lambda [$f $x]
    (match f math-expr
      {; symbol
       [,x 1]
       [?symbol? 0]
       ; function application
       [(,exp $g) (* (exp g) (∂/∂ g x))]
       [(,log $g) (* (/ 1 g) (∂/∂ g x))]
       [(,cos $g) (* (* -1 (sin g)) (∂/∂ g x))]
       [(,sin $g) (* (cos g) (∂/∂ g x))]
       [(,sqrt $g) (* (/ 1 (* 2 (sqrt g))) (∂/∂ g x))]
       [(,** $g $h) (* f (∂/∂ (* (log g) h) x))]
       [<apply $g $args>
        (sum (map 2#(* (capply `(add-user-script g %1) args) (∂/∂ %2 x))
                  (zip nats args)))]
       ; quote
       [<quote $g>
        (let {[$g' (∂/∂ g x)]}
          (if (monomial? g')
            g'
            (let {[$d (capply gcd (from-poly g'))]}
              (*' d '(map-poly (/' $ d) g')))))]
       ; term (constant)
       [,0 0]
       [(* _ ,1) 0]
       ; term (multiplication)
       [(* ,1 $fx^$n) (* n (** fx (- n 1)) (∂/∂ fx x))]
       [(* $a $fx^$n $r)
        (+ (* a (∂/∂ (**' fx n) x) r)
           (* a (**' fx n) (∂/∂ r x)))]
       ; polynomial
       [<poly $ts> (sum (map (∂/∂ $ x) ts))]
       ; quotient
       [(/ $p1 $p2)
        (let {[$p1' (∂/∂ p1 x)]
              [$p2' (∂/∂ p2 x)]}
          (/ (- (* p1' p2) (* p2' p1)) (** p2 2)))]
       })))
```

Demonstration of Differential Operator

```
(d/d (** x 2) x)
;(* 2 x)

(d/d (** a (** x 2)) x)
;(* 2 (** a x^2) (log a) x)

(d/d (* (cos x) (sin x)) x)
;(+ (* -1 (sin x)^2) (cos x)^2)

(d/d (/ 1 (+ 1 (exp (* -1 z)))) z)
;(/ (exp (* -1 z)) (+ 1 (* 2 (exp (* -1 z))) (exp (* -1 z))^2))

(d/d (d/d (log x) x) x)
;(/ -1 x^2)
```

Taylor Expansion

```
(define $taylor-expansion
  (lambda [$f $x $a]
    (multivariate-taylor-expansion f [| x |] [| a |])))

(define $multivariate-taylor-expansion
  (lambda [%f %xs %as]
    (with-symbols {h}
      (let {[$hs (generate-tensor 1#h_%1 (tensor-size xs))]}
        (map2 *
              (map 1#(/ 1 (fact %1)) nats0)
              (map (compose 1#(V.substitute xs as %1)
                            1#(V.substitute hs (with-symbols {i} (- xs_i as_i)) %1))
                   (iterate (compose 1#(∇ %1 xs) 1#(V.* hs %1)) f)))))))
```

Demonstration of Taylor Expansion

```
(take 4 (taylor-expansion (** e (* i x)) x 0))
;{1 (* i x) (/ (* -1 x^2) 2) (/ (* -1 i x^3) 6)}

(take 4 (taylor-expansion (* i (sin x)) x 0))
;{0 (* i x) 0 (/ (* -1 i x^3) 6)}

(take 4 (multivariate-taylor-expansion (** e (+ x y)) [| x y |] [| 0 0 |]))
;{1 (+ x y) (/ (+ x^2 (* 2 x y) y^2) 2) (/ (+ x^3 (* 3 x^2 y) (* 3 x y^2) y^3) 6)}

(take 3 (multivariate-taylor-expansion (f x y) [| x y |] [| 0 0 |]))
;{(f 0 0)
; (+ (* x (f|1 0 0)) (* y (f|2 0 0)))
; (/ (+ (* x^2 (f|1|1 0 0)) (* 2 x y (f|1|2 0 0)) (* y^2 (f|2|2 0 0))) 2)}
```

Features of Egison

Pattern-matching against the wider range of data types.

Customizable symbolic computation using Egison pattern-matching.

Tensor index notation in programming.

Tensor Index Notation in Egison

In Egison method, we can apply directly both "∂/∂" and "." functions to tensors.

$$R^{i}_{jkl} = \frac{\partial \Gamma^{i}_{jl}}{\partial x^{k}} - \frac{\partial \Gamma^{i}_{jk}}{\partial x^{l}} + \Gamma^{m}_{jl}\Gamma^{i}_{mk} - \Gamma^{m}_{jk}\Gamma^{i}_{ml}$$

Formula of Riemann curvature tensor

~: superscript
_: subscript

```
(define $R~i_j_k_l
  (with-symbols {m}
    (+ (- (∂/∂ Γ~i_j_l x~k) (∂/∂ Γ~i_j_k x~l))
       (- (. Γ~m_j_l Γ~i_m_k) (. Γ~m_j_k Γ~i_m_l)))))
```

Egison program that represents the above formula

Tensor Index Notation in Wolfram and Egison

$$R^{i}_{jkl} = \frac{\partial \Gamma^{i}_{jl}}{\partial x^{k}} - \frac{\partial \Gamma^{i}_{jk}}{\partial x^{l}} + \Gamma^{m}_{jl}\Gamma^{i}_{mk} - \Gamma^{m}_{jk}\Gamma^{i}_{ml}$$

Formula of Riemann curvature tensor

```
R=Table[D[Γ[[i,j,l]],x[[k]]] - D[Γ[[i,j,k]],x[[l]]]
        +Sum[Γ[[m,j,l]] Γ[[i,m,k]]
           - Γ[[m,j,k]] Γ[[i,m,l]],
             {m,M}],
         {i,M},{j,M},{k,M},{l,M}]
```

Wolfram program that represents the above formula

```
(define $R~i_j_k_l
  (with-symbols {m}
    (+ (- (∂/∂ Γ~i_j_l x~k) (∂/∂ Γ~i_j_k x~l))
       (- (. Γ~m_j_l Γ~i_m_k) (. Γ~m_j_k Γ~i_m_l)))))
```

Egison program that represents the above formula

The Reason for the Problems

There are two types of functions:

- Functions that should be mapped to each component of the tensors.

e.g. “+”, “-”, “*”, “/“, “∂/∂”, “min”, “max”, …

- Functions that should be applied directly to the tensors.

e.g. Tensor multiplication, matrix determinant, …

The existing work does not provide the easiest way to define both types of functions.

Proposed Method: Scalar and Tensor Parameters

When “$” is prepended to the beginning of the parameters, the function is applied to each component of tensors.

```
(define $min (lambda [$x $y] (if (less-than? x y) x y)))
```

Definition of the min function as the sample of scalar parameters

When “%” is prepended to the beginning of the parameters, the function treats the tensor argument as a whole.

```
(define $. (lambda [%t1 %t2] (contract + (* t1 t2))))
```

Definition of the “.” function as the sample of tensor parameters

Proposed Method: Scalar Parameters and Functions

```
(define $min (lambda [$x $y] (if (less-than? x y) x y)))
```

Definition of the min function as the sample of scalar parameters

$$min(\begin{pmatrix}1\\2\\3\end{pmatrix}_i, \begin{pmatrix}10\\20\\30\end{pmatrix}_j) = \begin{pmatrix}min(1,10) & min(1,20) & min(1,30)\\ min(2,10) & min(2,20) & min(2,30)\\ min(3,10) & min(3,20) & min(3,30)\end{pmatrix}_{ij} = \begin{pmatrix}1 & 1 & 1\\ 2 & 2 & 2\\ 3 & 3 & 3\end{pmatrix}_{ij}$$

Application of the min function to the vectors with different indices

$$min(\begin{pmatrix}1\\2\\3\end{pmatrix}_i, \begin{pmatrix}10\\20\\30\end{pmatrix}_i) = \begin{pmatrix}min(1,10) & min(1,20) & min(1,30)\\ min(2,10) & min(2,20) & min(2,30)\\ min(3,10) & min(3,20) & min(3,30)\end{pmatrix}_{ii} = \begin{pmatrix}min(1,10)\\ min(2,20)\\ min(3,30)\end{pmatrix}_i = \begin{pmatrix}1\\2\\3\end{pmatrix}_i$$

Application of the min function to the vectors with identical indices

Proposed Method: Tensor Parameters and Functions

```
(define $. (lambda [%t1 %t2] (contract + (* t1 t2))))
```

Definition of the "." function as the sample of scalar parameters

$$\begin{pmatrix}1\\2\\3\end{pmatrix}^{i} \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_{i} = contract(+, \begin{pmatrix}10 & 20 & 30\\20 & 40 & 60\\30 & 60 & 90\end{pmatrix}^{i}_{i}) = 10 + 40 + 90 = 140$$

$$\begin{pmatrix}1\\2\\3\end{pmatrix}_{i} \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_{i} = contract(+, \begin{pmatrix}10\\40\\90\end{pmatrix}_{i}) = \begin{pmatrix}10\\40\\90\end{pmatrix}_{i}$$

$$\begin{pmatrix}1\\2\\3\end{pmatrix}_{i} \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_{j} = contract(+, \begin{pmatrix}10 & 20 & 30\\20 & 40 & 60\\30 & 60 & 90\end{pmatrix}_{ij}) = \begin{pmatrix}10 & 20 & 30\\20 & 40 & 60\\30 & 60 & 90\end{pmatrix}_{ij}$$

Application of the "." function to vectors with various indices

Proposed Method: Implementation of Scalar Parameters

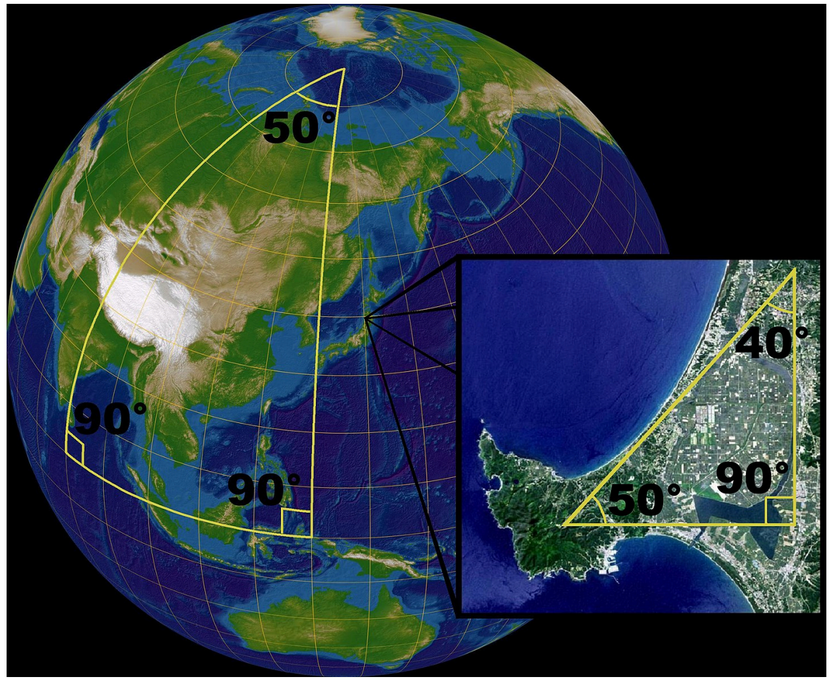
```
(lambda [$x $y] ...)
;=>(lambda [%x %y]
     (tensor-map (lambda [%x]
                   (tensor-map (lambda [%y] ...)
                               y))
                 x))
```

Simple implementation of the scalar and tensor parameters

Application: Riemannian Geometry

Calculation of Riemann Curvature of Spherical Surface

```
;; Coordinates for sphere
(define $x [|θ φ|])
(define $X [|(* r (sin θ) (cos φ)) ; = x
             (* r (sin θ) (sin φ)) ; = y
             (* r (cos θ))         ; = z
             |])

;; Local basis
(define $e ((flip ∂/∂) x~# X_#))

;; Metric tensor
(define $g__ (generate-tensor 2#(V.* e_%1 e_%2) {2 2}))
(define $g~~ (M.inverse g_#_#))

;; Christoffel symbols of the first kind
(define $Γ_i_j_k
  (* (/ 1 2)
     (+ (∂/∂ g_i_j x_k)
        (∂/∂ g_i_k x_j)
        (* -1 (∂/∂ g_j_k x_i)))))

;; Christoffel symbols of the second kind
(define $Γ~__ (with-symbols {i} (. g~#~i Γ_i_#_#)))

;; Riemann curvature tensor
(define $R~i_j_k_l
  (with-symbols {m}
    (+ (- (∂/∂ Γ~i_j_l x_k) (∂/∂ Γ~i_j_k x_l))
       (- (. Γ~m_j_l Γ~i_m_k) (. Γ~m_j_k Γ~i_m_l)))))
```



https://commons.wikimedia.org/wiki/File:Triangles_(spherical_geometry).jpg

https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

Local Basis - Spherical Surface

```
;;;
;;; Parameters
;;;

(define $x [|θ φ|])

(define $X [|(* r (sin θ) (cos φ)) ; = x
             (* r (sin θ) (sin φ)) ; = y
             (* r (cos θ))         ; = z
             |])

;;
;; Local basis
;;

(define $e ((flip ∂/∂) x~# X_#))
e
;[|[|(* r (cos θ) (cos φ)) (* r (cos θ) (sin φ)) (* -1 r (sin θ)) |]
;  [|(* -1 r (sin θ) (sin φ)) (* r (sin θ) (cos φ)) 0 |]
;  |]_#~#
```


https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

Riemann Metrics - Spherical Surface

```
;;
;; Metric tensor
;;

(define $g__ (generate-tensor 2#(V.* e_%1 e_%2) {2 2}))
(define $g~~ (M.inverse g_#_#))

g_#_#;[| [| r^2 0 |] [| 0 (* r^2 (sin θ)^2) |] |]_#_#
g~#~#;[| [| (/ 1 r^2) 0 |] [| 0 (/ 1 (* r^2 (sin θ)^2)) |] |]~#~#
```

Rakuten

https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

Christoffel Symbols of the First Kind - Spherical Surface

$$\Gamma_{ijk} = \frac{1}{2}\left(\frac{\partial g_{ij}}{\partial x^k} + \frac{\partial g_{ik}}{\partial x^j} - \frac{\partial g_{kj}}{\partial x^i}\right)$$

```
;;
;; Christoffel symbols of the first kind
;;

(define $Γ_j_k_l
  (* (/ 1 2)
     (+ (∂/∂ g_j_l x_k)
        (∂/∂ g_j_k x_l)
        (* -1 (∂/∂ g_k_l x_j)))))

Γ_1_#_#;[| [| 0 0 |] [| 0 (* -1 r^2 (sin θ) (cos θ)) |] |]_#_#
Γ_2_#_#;[| [| 0 (* r^2 (sin θ) (cos θ)) |] [| (* r^2 (sin θ) (cos θ)) 0 |] |]_#_#
```


https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

Christoffel Symbols of the Second Kind - Spherical Surface

$$\Gamma^{i}_{kl} = g^{ij}\Gamma_{jkl}$$

```
;;
;; Christoffel symbols of the second kind
;;

(define $Γ~__ (with-symbols {i} (. g~#~i Γ_i_#_#)))

Γ~1_#_#;[| [| 0 0 |] [| 0 (* -1 (sin θ) (cos θ)) |] |]_#_#
Γ~2_#_#;[| [| 0 (/ (cos θ) (sin θ)) |] [| (/ (cos θ) (sin θ)) 0 |] |]_#_#
```


https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

Riemann Curvature Tensor - Spherical Surface

$$R^i_{jkl} = \frac{\partial \Gamma^i_{jl}}{\partial x^k} - \frac{\partial \Gamma^i_{jk}}{\partial x^l} + \Gamma^m_{jl}\Gamma^i_{mk} - \Gamma^m_{jk}\Gamma^i_{ml}$$

```
;;
;; Riemann curvature tensor
;;

(define $R~i_j_k_l
  (with-symbols {m}
    (+ (- (∂/∂ Γ~i_j_l x_k) (∂/∂ Γ~i_j_k x_l))
       (- (. Γ~m_j_l Γ~i_m_k) (. Γ~m_j_k Γ~i_m_l)))))

R~#_#_1_1;[| [| 0 0 |] [| 0 0 |] |]~#_#
R~#_#_1_2;[| [| 0 (* -1 (sin θ)^2) |] [| 1 0 |] |]~#_#
R~#_#_2_1;[| [| 0 (sin θ)^2 |] [| -1 0 |] |]~#_#
R~#_#_2_2;[| [| 0 0 |] [| 0 0 |] |]~#_#
```


https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

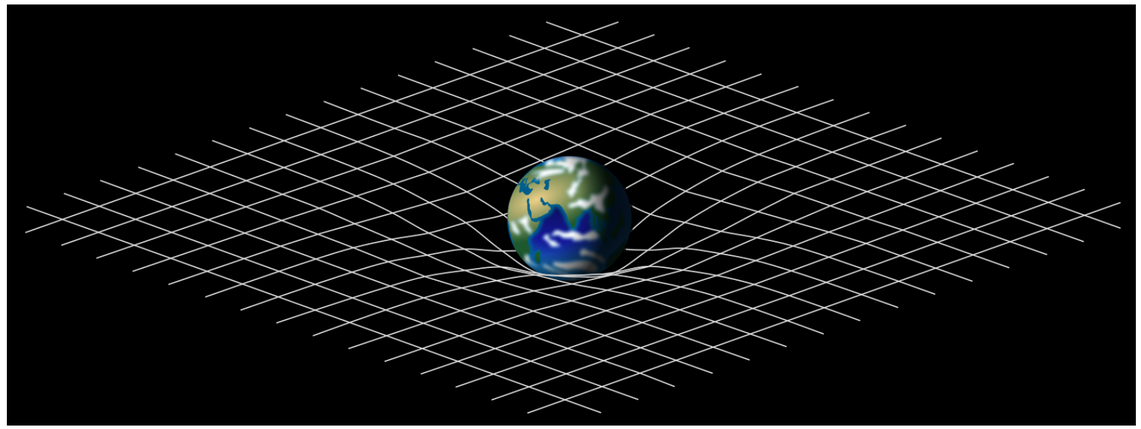
Ricci and Scalar Curvature - Spherical Surface

```
;;
;; Ricci curvature
;;

(define $Ric__ (with-symbols {i} (contract + R~i_#_i_#)))

Ric_#_#;[| [| 1 0 |] [| 0 (sin θ)^2 |] |]_#_#

;;
;; Scalar curvature
;;

(define $scalar-curvature (with-symbols {j k} (. g~j~k Ric_j_k)))

scalar-curvature;(/ 2 r^2)
```


https://www.egison.org/math/riemann-curvature-tensor-of-S2.html

Calculation of Riemann Curvature of Schwarzschild Space

```
;; Parameters
(define $x [|t r θ φ|])

;; Metric tensor
(define $g__
  [|[| (/ '(- (* c^2 r) (* 2 G M)) (* c^2 r)) 0 0 0 |]
    [| 0 (/ -1 (/ '(- (* c^2 r) (* 2 G M)) (* c^2 r))) 0 0 |]
    [| 0 0 (* -1 r^2) 0 |]
    [| 0 0 0 (* -1 r^2 (sin θ)^2) |]
    |])

(define $g~~ (M.inverse g_#_#))

;; Christoffel symbols of the first kind
(define $Γ_j_k_l
  (* (/ 1 2)
     (+ (∂/∂ g_j_k x_l)
        (∂/∂ g_j_l x_k)
        (* -1 (∂/∂ g_k_l x_j)))))

;; Christoffel symbols of the second kind
(define $Γ~__ (with-symbols {i} (. g~#~i Γ_i_#_#)))

;; Riemann curvature tensor
(define $R~i_j_k_l
  (with-symbols {m}
    (expand-all (+ (- (∂/∂ Γ~i_j_l x_k) (∂/∂ Γ~i_j_k x_l))
                   (- (. Γ~m_j_l Γ~i_m_k) (. Γ~m_j_k Γ~i_m_l))))))
```



https://commons.wikimedia.org/wiki/File:Spacetime_lattice_analogy.svg

Egison Paper on Tensor Index Notation

Scalar and Tensor Parameters for Importing Tensor Index Notation including Einstein Summation Notation

SATOSHI EGI, Rakuten Institute of Technology

In this paper, we propose a method for importing tensor index notation, including Einstein summation notation, into functional programming. This method involves introducing two types of parameters, i.e, scalar and tensor parameters, and simplified tensor index rules that do not handle expressions that are valid only for the Cartesian coordinate system, in which the index can move up and down freely. An example of such an expression is "$c = A_i B_i$". As an ordinary function, when a tensor parameter obtains a tensor as an argument, the function treats the tensor argument as a whole. In contrast, when a scalar parameter obtains a tensor as an argument, the function is applied to each component of the tensor. In this paper, we show that introducing these two types of parameters and our simplified index rules enables us to apply arbitrary user-defined functions to tensor arguments using index notation including Einstein summation notation without requiring an additional description to enable each function to handle tensors.

CCS Concepts: • **Software and its engineering → General programming languages**; • **Mathematics of computing → Computations on matrices**;

Additional Key Words and Phrases: tensor, index notation, Einstein summation notation, scalar parameters, tensor parameters, scalar functions, tensor functions

ACM Reference format:
Satoshi Egi. 2017. Scalar and Tensor Parameters for Importing Tensor Index Notation including Einstein Summation Notation. 1, 1, Article 1 (August 2017), 18 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Tensor analysis is one of the fields of mathematics in which we can easily find notations that have not been imported into popular programming languages [6, 13]. Index notation is one such notation widely used by mathematicians to describe expressions in tensor analysis concisely. This paper proposes a method for importing it into programming.

Tensor analysis is also a field with a wide range of application. For example, the general theory of relativity is formulated in terms of tensor analysis. In addition, tensor analysis plays an important role in other theories in physics, such as fluid dynamics. In fields more familiar to computer scientists, tensor analysis is necessary for computer vision [7]. Tensor analysis also appears in the theory of machine learning to handle multidimensional data. The importance of tensor calculation is increasing day by day even in computer science.

Concise notation for tensor calculation in programming will simplify technical programming in many areas. Therefore, it is important to develop a method for describing tensor calculation concisely in programs.

XXXX-XXXX/2017/8-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

https://arxiv.org/abs/1702.06343

Differential Forms

Differential Forms in Egison

We can define and use the operators for differential forms concisely in Egison.

$$R^i_{jkl} = \frac{\partial \Gamma^i_{jl}}{\partial x^k} - \frac{\partial \Gamma^i_{jk}}{\partial x^l} + \Gamma^m_{jl}\Gamma^i_{mk} - \Gamma^m_{jk}\Gamma^i_{ml} \quad \Leftrightarrow \quad \Omega^i_j = d\omega^i_j + \omega^i_k \wedge \omega^k_j$$

Formula of Riemann curvature

Formula of curvature form

```
(define $Ω
  (with-symbols {i j}
    (+ (d ω~i_j) (wedge ω~i_k ω~k_j))))
```

Egison program that represents the formula of curvature form

Differential Forms in Egison

We achieved that by providing users the method for controlling the completion of omitted indices.

$$R^i_{jkl} = \frac{\partial \Gamma^i_{jl}}{\partial x^k} - \frac{\partial \Gamma^i_{jk}}{\partial x^l} + \Gamma^m_{jl}\Gamma^i_{mk} - \Gamma^m_{jk}\Gamma^i_{ml} \quad \Leftrightarrow \quad \Omega^i_j = d\omega^i_j + \omega^i_k \wedge \omega^k_j$$

Formula of Riemann curvature

Formula of curvature form

```
(+ A B)
;(+ A_t1_t2 B_t1_t2)

!(+ A B)
;(+ A_t1_t2 B_t3_t4)
```

By default, the same indices are completed to each tensor of the arguments.

When "!" is prepended to the function application, the different indices are completed to each tensor of the arguments.

Wedge Product, Exterior Derivative, Hodge operator, and Codifferential Operator

```
(define $wedge
  (lambda [%X %Y]
    !(. X Y)))

(define $d
  (lambda [%A]
    !((flip ∂/∂) x A)))

(define $hodge
  (lambda [%A]
    (let {[$k (df-order A)]}
      (with-symbols {i j}
        (* (sqrt (abs (M.det g_#_#)))
           (foldl . (. (subrefs A (map 1#j_%1 (between 1 k)))
                       (subrefs (ε' N k) (map 1#i_%1 (between 1 N))))
                  (map 1#g~[i_%1]~[j_%1] (between 1 k))))))))

(define $δ
  (lambda [%A]
    (let {[$r (df-order A)]}
      (* (** -1 (+ (* N r) 1))
         (hodge (d (hodge A)))))))
```

Wedge Product - Euclid Space

```
(define $N 3)
(define $params [| x y z |])
(define $g [| [| 1 0 0 |] [| 0 1 0 |] [| 0 0 1 |] |])

(define $wedge
  (lambda [%X %Y]
    !(. X Y)))

(define $dx [| 1 0 0 |])
(define $dy [| 0 1 0 |])
(define $dz [| 0 0 1 |])

(wedge dx dy)
;[| [| 0 1 0 |] [| 0 0 0 |] [| 0 0 0 |] |]

(df-normalize (wedge dx dy))
;[| [| 0 (/ 1 2) 0 |] [| (/ -1 2) 0 0 |] [| 0 0 0 |] |]

(wedge dz dz)
;[| [| 0 0 0 |] [| 0 0 0 |] [| 0 0 1 |] |]

(df-normalize (wedge dz dz))
;[| [| 0 0 0 |] [| 0 0 0 |] [| 0 0 0 |] |]
```


https://www.egison.org/math/wedge-product.html

Exterior Derivative - Euclid Space

```
(define $N 3)
(define $params [| x y z |])
(define $g [| [| 1 0 0 |] [| 0 1 0 |] [| 0 0 1 |] |])

(define $d
  (lambda [%X]
    !((flip ∂/∂) params X)))

(d (f x y z))
;[| (f|1 x y z) (f|2 x y z) (f|3 x y z) |]

(df-normalize (d (d (f x y z))))
;[| [| 0 0 0 |] [| 0 0 0 |] [| 0 0 0 |] |]
```

Rakuten

https://www.egison.org/math/exterior-derivative.html

Hodge Operator - Euclid Space

```
(define $N 3)
(define $params [| x y z |])
(define $g [| [| 1 0 0 |] [| 0 1 0 |] [| 0 0 1 |] |])
```

```
(define $hodge
  (lambda [%A]
    (let {[$k (df-order A)]}
      (with-symbols {i j}
        (* (sqrt (abs (M.det g_#_#)))
           (foldl . (. (subrefs A (map 1#j_%1 (between 1 k)))
                       (subrefs (ε' N k) (map 1#i_%1 (between 1 N))))
                  (map 1#g~[i_%1]~[j_%1] (between 1 k))))))))
```

```
(define $dx [| 1 0 0 |])
(define $dy [| 0 1 0 |])
(define $dz [| 0 0 1 |])
```

$$\star\,\alpha = \frac{1}{k!(n-k)!}\epsilon_{i_1,\ldots,i_n}\sqrt{|\det(g)|}\,\alpha_{j_1,\ldots j_k}g^{i_1 j_1}\cdots g^{i_k j_k}e^{i_{k+1}}\wedge\cdots\wedge e^{i_n}$$

https://ncatlab.org/nlab/show/Hodge+star+operator

```
(hodge dx)
;[| [| 0 0 0 |] [| 0 0 1 |] [| 0 0 0 |] |] = (wedge dy dz)

(hodge (wedge dx dy))
;[| 0 0 1 |] = dz
```

https://www.egison.org/math/hodge-E3.html

Hodge Operator - Minkowski Space

```
(define $N 4)
(define $params [| t x y z |])
(define $g [| [| -1 0 0 0 |] [| 0 1 0 0 |] [| 0 0 1 0 |] [| 0 0 0 1 |] |])

(define $hodge
  (lambda [%A]
    (let {[$k (df-order A)]}
      (with-symbols {i j}
        (* (sqrt (abs (M.det g_#_#)))
           (foldl . (. (subrefs A (map 1#j_%1 (between 1 k)))
                       (subrefs (ε' N k) (map 1#i_%1 (between 1 N))))
                  (map 1#g~[i_%1]~[j_%1] (between 1 k))))))))

(define $dt [| 1 0 0 0 |])
(define $dx [| 0 1 0 0 |])
(define $dy [| 0 0 1 0 |])
(define $dz [| 0 0 0 1 |])

(hodge (wedge dt dx))
;[| [| 0 0 0 0 |] [| 0 0 0 0 |] [| 0 0 0 -1 |] [| 0 0 0 0 |] |] = (* -1 (wedge dy dz))

(hodge (wedge dy dz))
;[| [| 0 1 0 0 |] [| 0 0 0 0 |] [| 0 0 0 0 |] [| 0 0 0 0 |] |] = (wedge dt dx)
```

Codifferential Operator - Euclid Space

```
(define $N 3)

(define $g [| [| 1 0 0 |] [| 0 1 0 |] [| 0 0 1 |] |])

(define $δ
  (lambda [%A]
    (let {[$r (df-order A)]}
      (* (** -1 (+ (* N r) 1))
         (hodge (d (hodge A)))))))

(δ [| (* x 2) (* y 2) (* z 2) |])
;6
```

Application: Geometry of Differential Forms

Hodge Laplacian

$$\Delta = d\delta + \delta d$$

$$u_{xx} + u_{yy} = u_{rr} + \left(\frac{1}{r}\right)u_r + \left(\frac{1}{r^2}\right)u_{\theta\theta}$$

```
(define $N 2)

(define $x [|r θ|])

(define $g__ [| [| 1 0 |] [| 0 r^2 |] |])
(define $g~~ (M.inverse g_#_#))

(define $Δ
  (lambda [%A]
    (match (df-order A) integer
      {[,0 (δ (d A))]
       [,2 (d (δ A))]
       [_ (+ (d (δ A)) (δ (d A)))]})))

(Δ (f r θ))
;(/ (+ (* -1 (f|2|2 r θ)) (* -1 r (f|1 r θ)) (* -1 r^2 (f|1|1 r θ))) r^2)
```

Rakuten

https://www.egison.org/math/hodge-laplacian-polar.html

Euler Form

```
(define $ω0 Γ~#_#_#)

(define $A [|[| (/ 1 r) 0 |] [| 0 (/ 1 (* r (sin θ))) |]|])

(define $ω (+ (. (M.inverse A)~i_j ω0~j_k A~k_l) (. (M.inverse A)~i_j (d A~j_l))))

;;; Curvature form
(define $Ω'
  (with-symbols {i j}
    (+ (d ω~i_j)
       (wedge ω~i_k ω~k_j))))

(define $Ω
  (with-symbols {i j t1 t2}
    (- Ω'~i_j_t1_t2 Ω'~i_j_t2_t1)))

;;; Euler form
(define $euler-form (* (/ 1 (* 4 π)) (- Ω~1_2 Ω~2_1)))
euler-form;[| [| 0 (/ (sin θ) (* 2 π)) |] [| (/ (* -1 (sin θ)) (* 2 π)) 0 |] |]

; χ(S^2) = ∫ dθ dφ (/ (sin θ) (* 2 π)) = ∫ dθ (sin θ)
; = [ (* -1 (cos θ)) ] 0-π = (cos 0) - (cos π) = 2
```

Rakuten

https://www.egison.org/math/euler-form-of-S2.html

Euler Form

```
(define $ω0 Γ~#_#_#)

(define $A [|[| (/ 1 r) 0 |] [| 0 (/ 1 (* r (sin θ))) |]|])

(define $ω (+ (. (M.inverse A)~i_j ω0~j_k A~k_l) (. (M.inverse A)~i_j (d A~j_l))))

;;; Curvature form
(define $Ω'
  (with-symbols {i j}
    (+ (d ω~i_j)
       (wedge ω~i_k ω~k_j))))

(define $Ω
  (with-symbols {i j t1 t2}
    (- Ω'~i_j_t1_t2 Ω'~i_j_t2_t1)))

;;; Euler form
(define $euler-form (* (/ 1 (* 4 π)) (- Ω~1_2 Ω~2_1)))
euler-form;[| [| 0 (/ (sin θ) (* 2 π)) |] [| (/ (* -1 (sin θ)) (* 2 π)) 0 |] |]

; χ(S^2) = ∫ dθ dϕ (/ (sin θ) (* 2 π)) = ∫ dθ (sin θ)
; = [ (* -1 (cos θ)) ] 0-π = (cos 0) - (cos π) = 2
```

$$\omega = a^{-1}\omega_0 a + a^{-1}da$$

U(1) Yang-Mills Equation

```
(define $F [|[| 0 (Ex t x y z) (Ey t x y z) (Ez t x y z) |]
              [| (* -1 (Ex t x y z)) 0 (Bz t x y z) (* -1 (By t x y z)) |]
              [| (* -1 (Ey t x y z)) (* -1 (Bz t x y z)) 0 (Bx t x y z) |]
              [| (* -1 (Ez t x y z)) (By t x y z) (* -1 (Bx t x y z)) 0 |]|])

(hodge (d F))
;[|(+ (* -2 (Bz|4 t x y z)) (* -2 (By|3 t x y z)) (* -2 (Bx|2 t x y z)))
;  (+ (* -2 (Ey|4 t x y z)) (* 2 (Ez|3 t x y z)) (* -2 (Bx|1 t x y z)))
;  (+ (* 2 (Ex|4 t x y z)) (* -2 (Ez|2 t x y z)) (* -2 (By|1 t x y z)))
;  (+ (* -2 (Ex|3 t x y z)) (* 2 (Ey|2 t x y z)) (* -2 (Bz|1 t x y z)))|]

;(∇ B) = 0, (rot x E) = ∂t B, (rot y E) = ∂t B, (rot z E) = ∂t B

(δ F)
;[|(+ (* -2 (Ez|4 t x y z)) (* -2 (Ey|3 t x y z)) (* -2 (Ex|2 t x y z)))
;  (+ (* 2 (By|4 t x y z)) (* -2 (Bz|3 t x y z)) (* -2 (Ex|1 t x y z)))
;  (+ (* -2 (Bx|4 t x y z)) (* 2 (Bz|2 t x y z)) (* -2 (Ey|1 t x y z)))
;  (+ (* 2 (Bx|3 t x y z)) (* -2 (By|2 t x y z)) (* -2 (Ez|1 t x y z)))|]

;(∇ E) = 0, (rot x B) = ∂t E, (rot y B) = ∂t E, (rot z B) = ∂t E
```

Egison Mathematics Notebook

Number Theory

Elementary Number Theory	Tribonacci Number Euler's Totient Function
Root of Unity	5th Root of Unity 7th Root of Unity 9th Root of Unity 17th Root of Unity
Quadratic Field	Gaussian Primes Eisenstein Primes

Algebra

Algebraic Equation	Quadratic Equation Cubic Equation Quartic Equation
Root of Unity	5th Root of Unity 7th Root of Unity 9th Root of Unity 17th Root of Unity

Mathematical Analysis

Derivative	Laplacian in Polar Coordinates Laplacian in Spherical Coordinates
Series	Euler's Formula
Fourier Analysis	Leibniz Formula

Geometry

Riemannian Geometry	Gaussian Curvature Riemann Curvature Tensor of S^2 Riemann Curvature Tensor of S^3 Riemann Curvature Tensor of S^4 Riemann Curvature Tensor of S^5 Riemann Curvature Tensor of S^7 Riemann Curvature Tensor of T^2 Riemann Curvature Tensor of $S^2 \times S^3$ Schwarzschild Metric Friedmann–Lemaître–Robertson–Walker Metric
Differential Forms	Wedge Product Exterior Derivative Hodge Operator of E^3 Hodge Operator of Minkowski Space Hodge Laplacian of Polar Coordinates Hodge Laplacian of Spherical Coordinates Curvature Form
Application to Physics	Vector Analysis Yang-Mills Equation of $U(1)$ Gauge Theory
Characteristic Classes	Euler Form of S^2 Euler Form of T^2

Future Work

Combine Egison with Formura.

- https://github.com/formura/formura

- If we achieved that, we can execute differential equations using the language of differential forms on super computers.

Collaboration with Fukagawa-san.

- https://twitter.com/hiroki_f/status/930018952654200832

Table of Contents

Aim of Egison

My aim is to create a language that can represent directly all algorithms that can be discovered.

Currently, my biggest challenge is to improve Egison in order to represent directly calculations that appear in mathematical physics.

Research of Science

Nature

Understanding of nature develops our intuition.

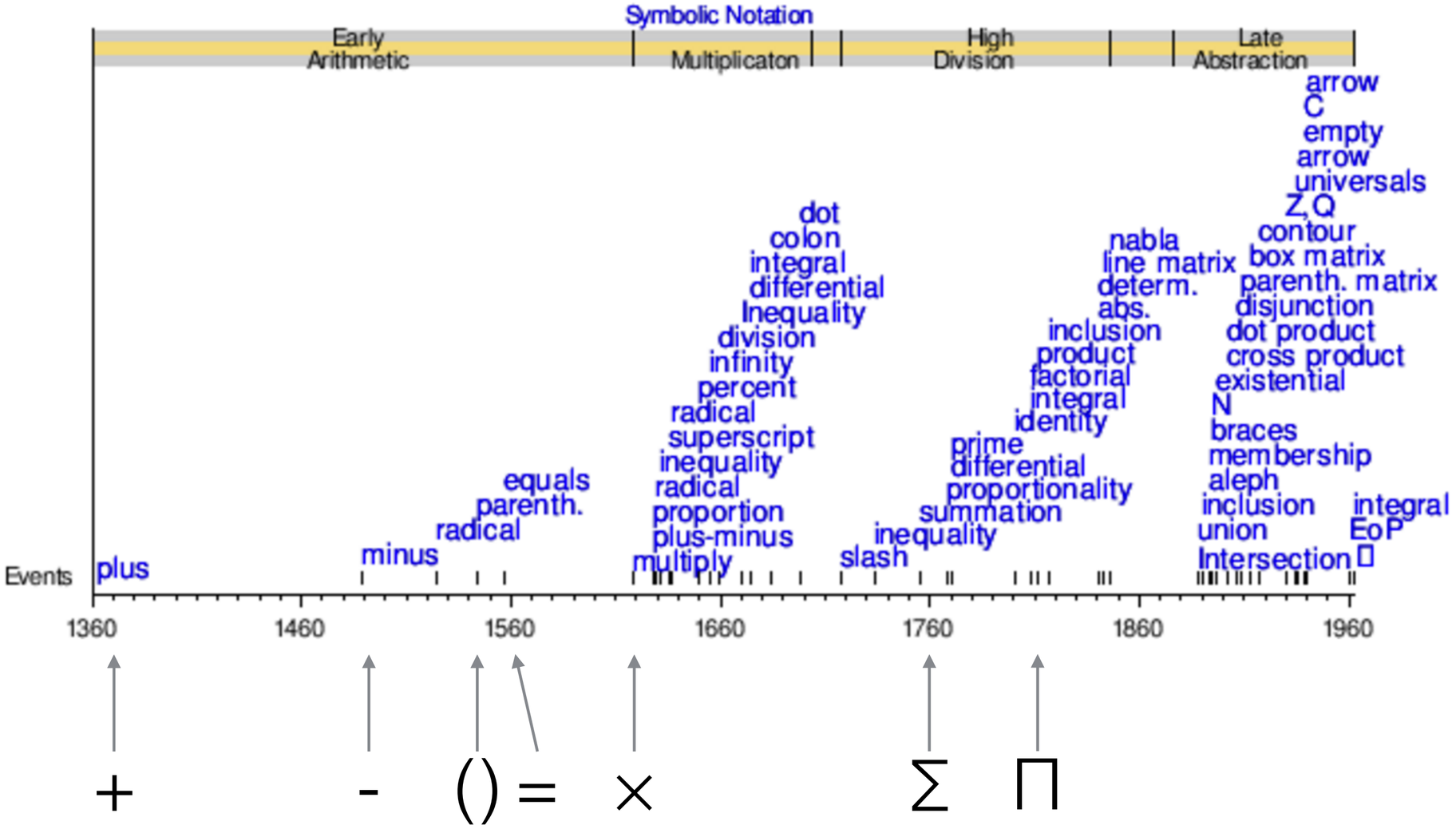
Good representation promotes us to deepen our understanding of nature.

Intuition

Representation

Better representation is invented to express developed intuition.

History of Mathematical Symbols



https://en.wikipedia.org/wiki/History_of_mathematical_notation

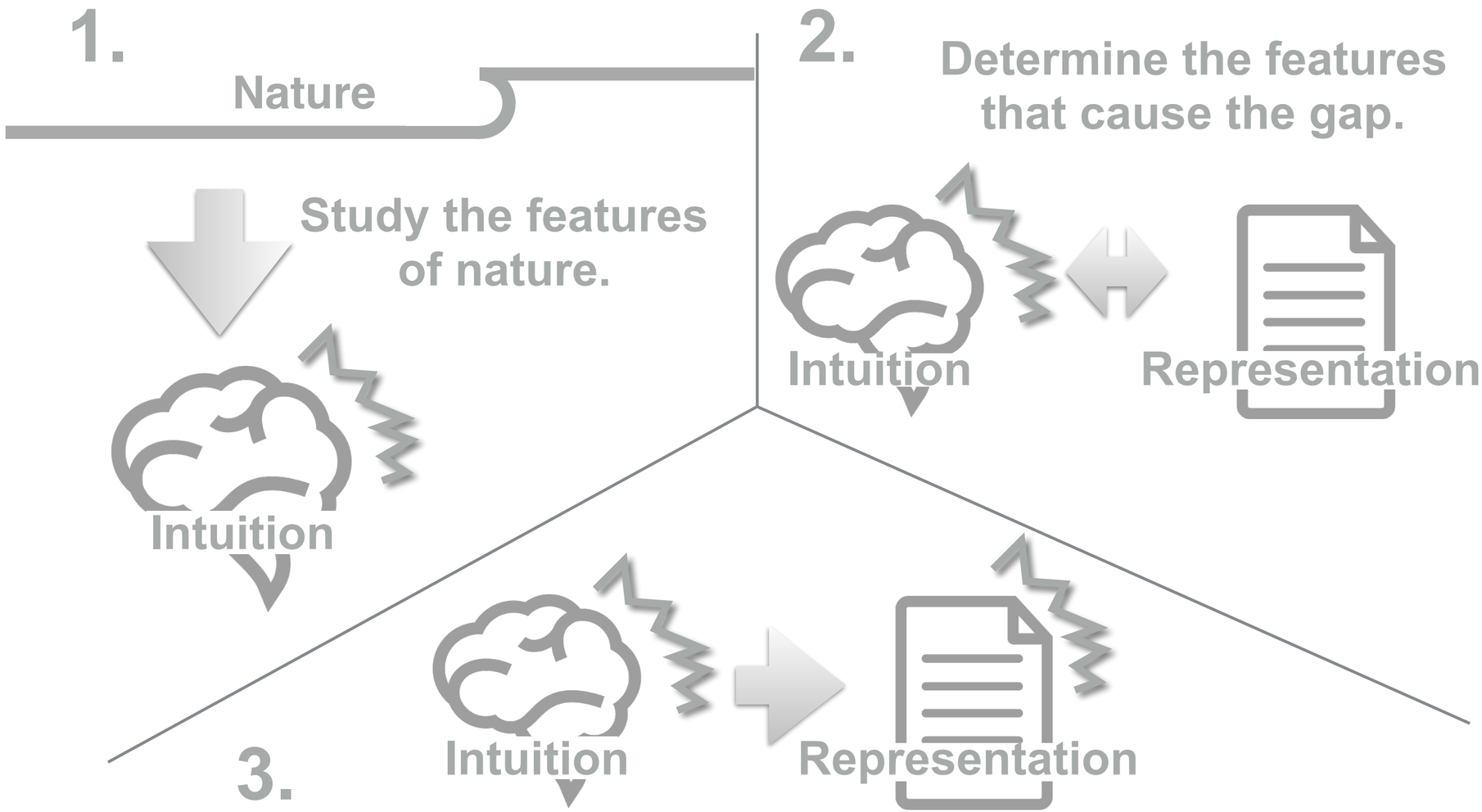
Number Notations

MMMCCCCVIIII

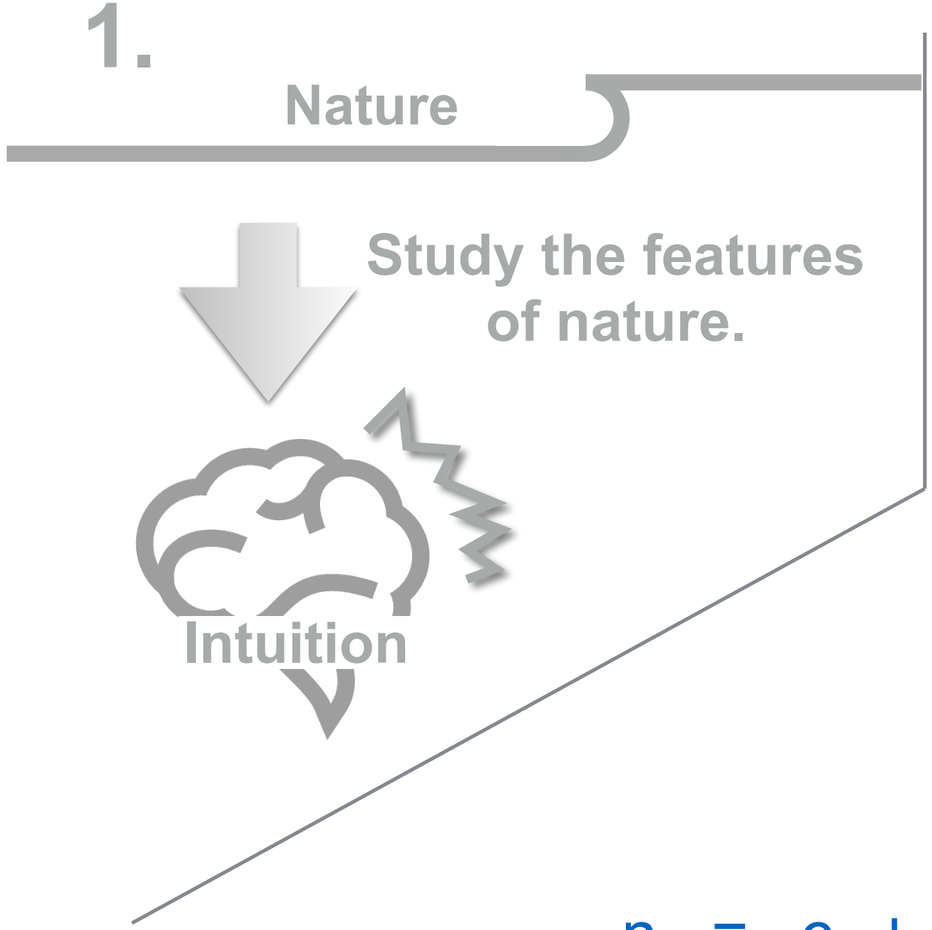
三千四百九

3409

The Process of Inventing New Representation



The Process of Inventing New Representation



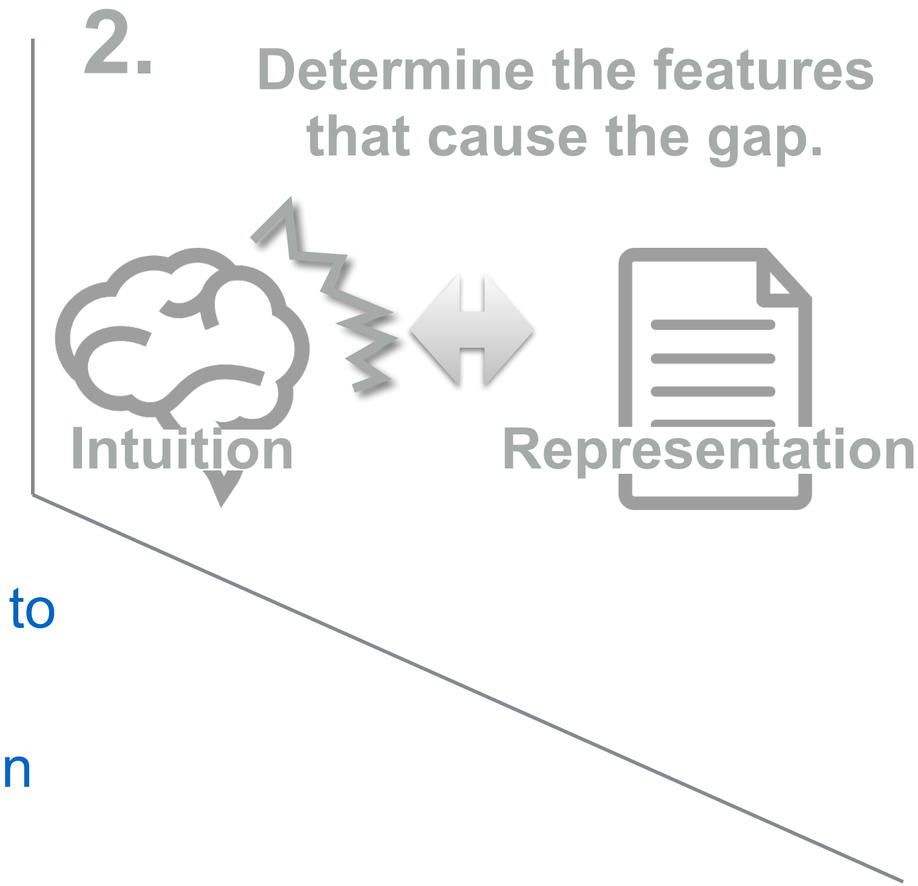
Example (Number notation):

- Discovery of numbers.
- Discovery of addition.
- Discovery of multiplication.

Discovery of the rule that all numbers are uniquely represented in the following form.

$$n = c_0 + c_1 \cdot 10^1 + c_2 \cdot 10^2 + \ldots + c_r \cdot 10^r$$

The Process of Inventing New Representation



Example (Number notation):

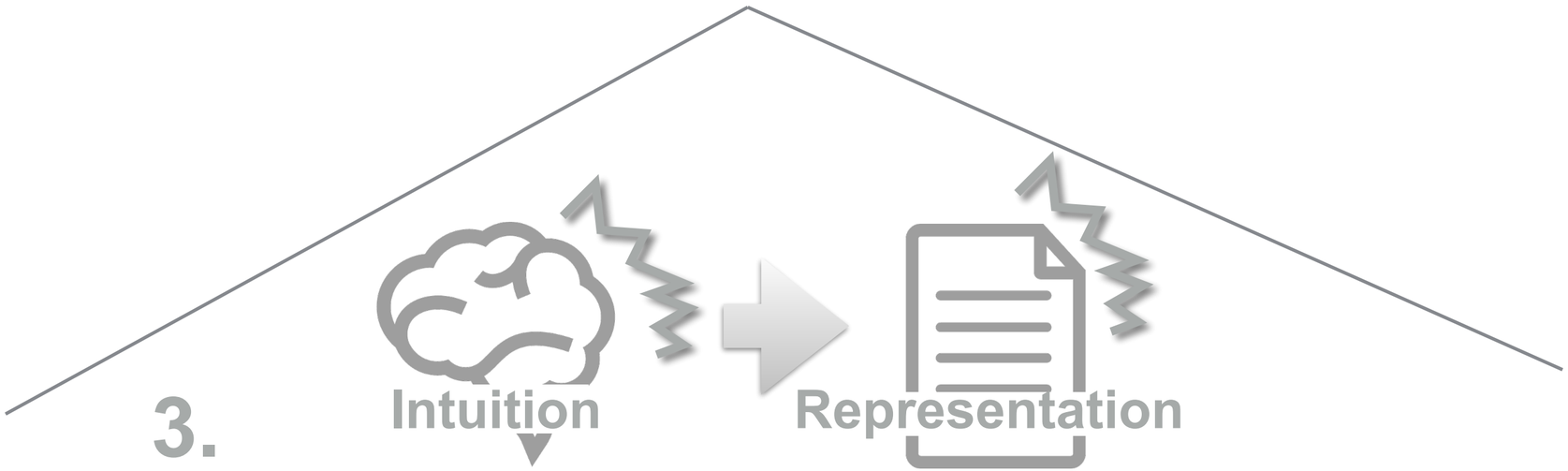
- Discovery that the following rule is useful to improve representation.
  - All numbers are uniquely represented in the following form.

$$n = c_0 + c_1 \cdot 10^1 + c_2 \cdot 10^2 + \ldots + c_r \cdot 10^r$$

The Process of Inventing New Representation

Example (Number notation):

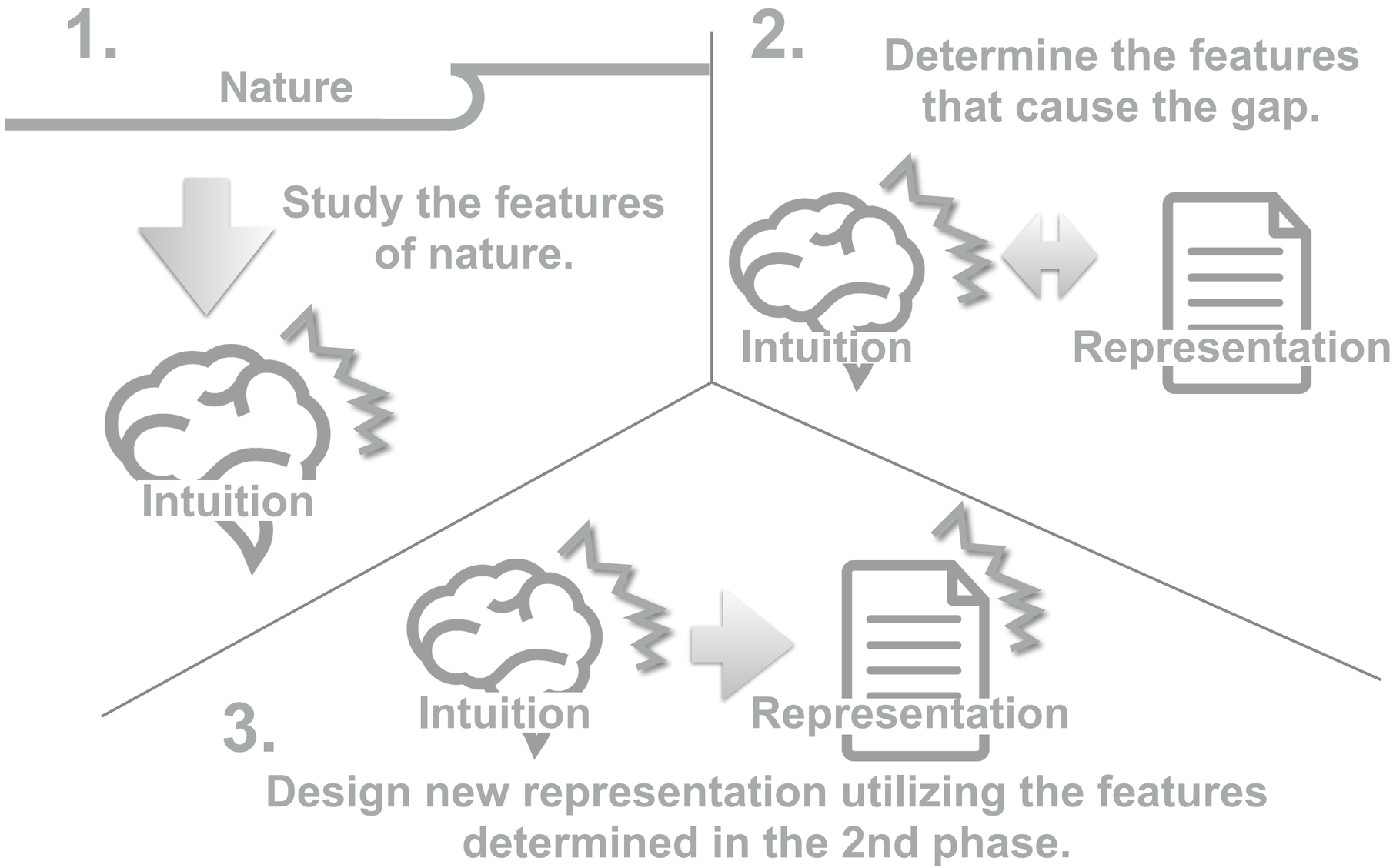
- Idea to give a name to each number.
- Invention of the decimal numeral system.
- Invention of ‘0’ as a placeholder.



3. Design new representation utilizing the features determined in the 2nd phase.

Rakuten

The Process of Inventing New Representation



Grand Questions

Is there a method for measuring the expressive power of languages?

Is there a common method for finding the better representation?

Table of Contents

Introduction of the features of Egison.

Philosophy behind the creation of Egison and future direction.

Demonstration.

THANK YOU

Rakuten