パターンマッチ指向プログラミング言語Egisonとその上に実装された数式処理システム

Sep 16, 2018

Satoshi Egi
Rakuten Institute of Technology
Rakuten, Inc.

Rakuten

本日の発表の流れ

Egisonの概要

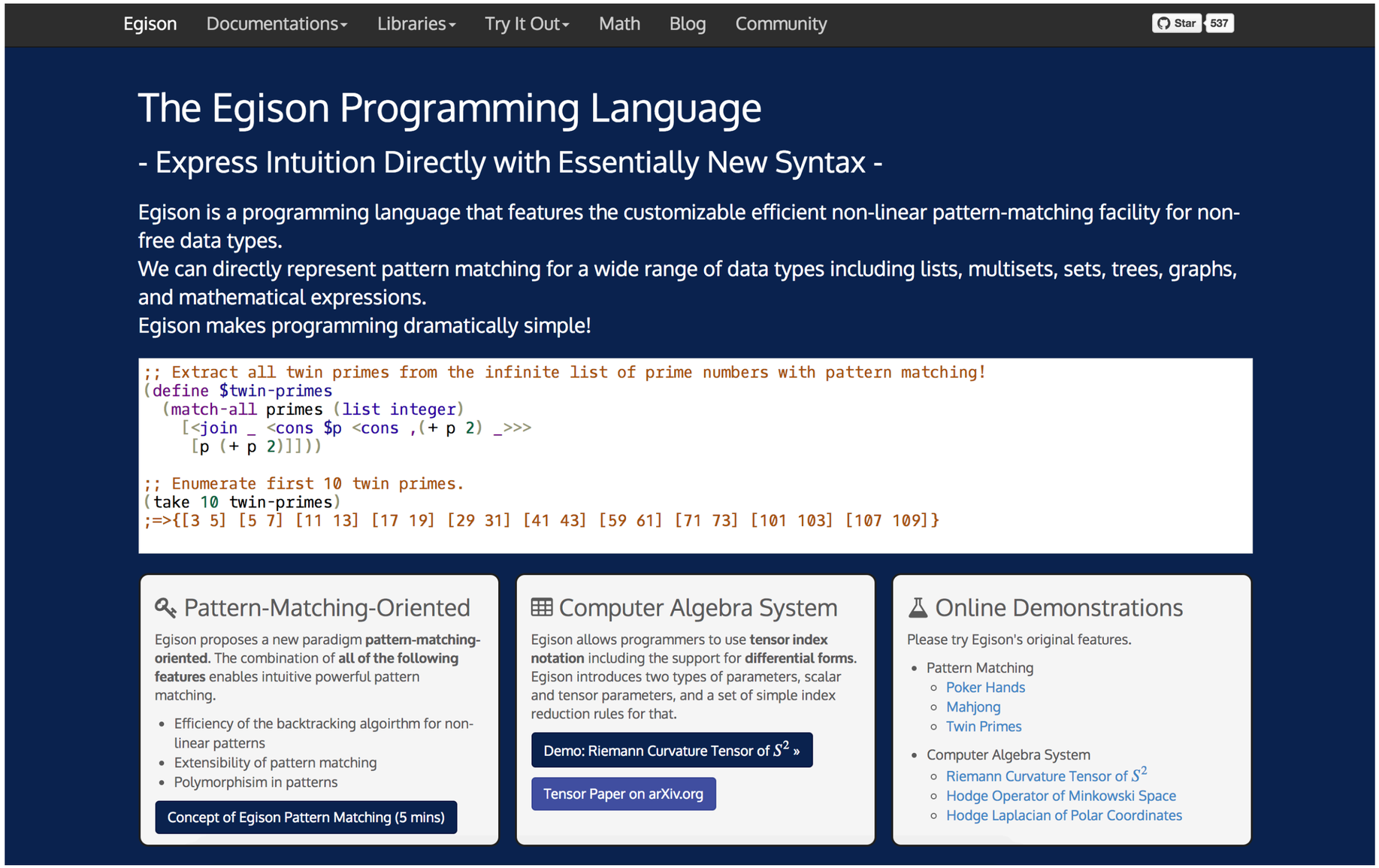
Egison独自の機能

本日の発表の流れ

Egisonの概要

Egison独自の機能

Egisonのウェブサイト



https://www.egison.org

Egisonのインストール

Macへのインストールは特に簡単.

$ brew update

$ brew tap egison/egison

$ brew install egison

Linux, Windowsへのインストールも簡単.

Haskell Platformをインストールしたあとに, 以下のコマンドを実行.

$ cabal update

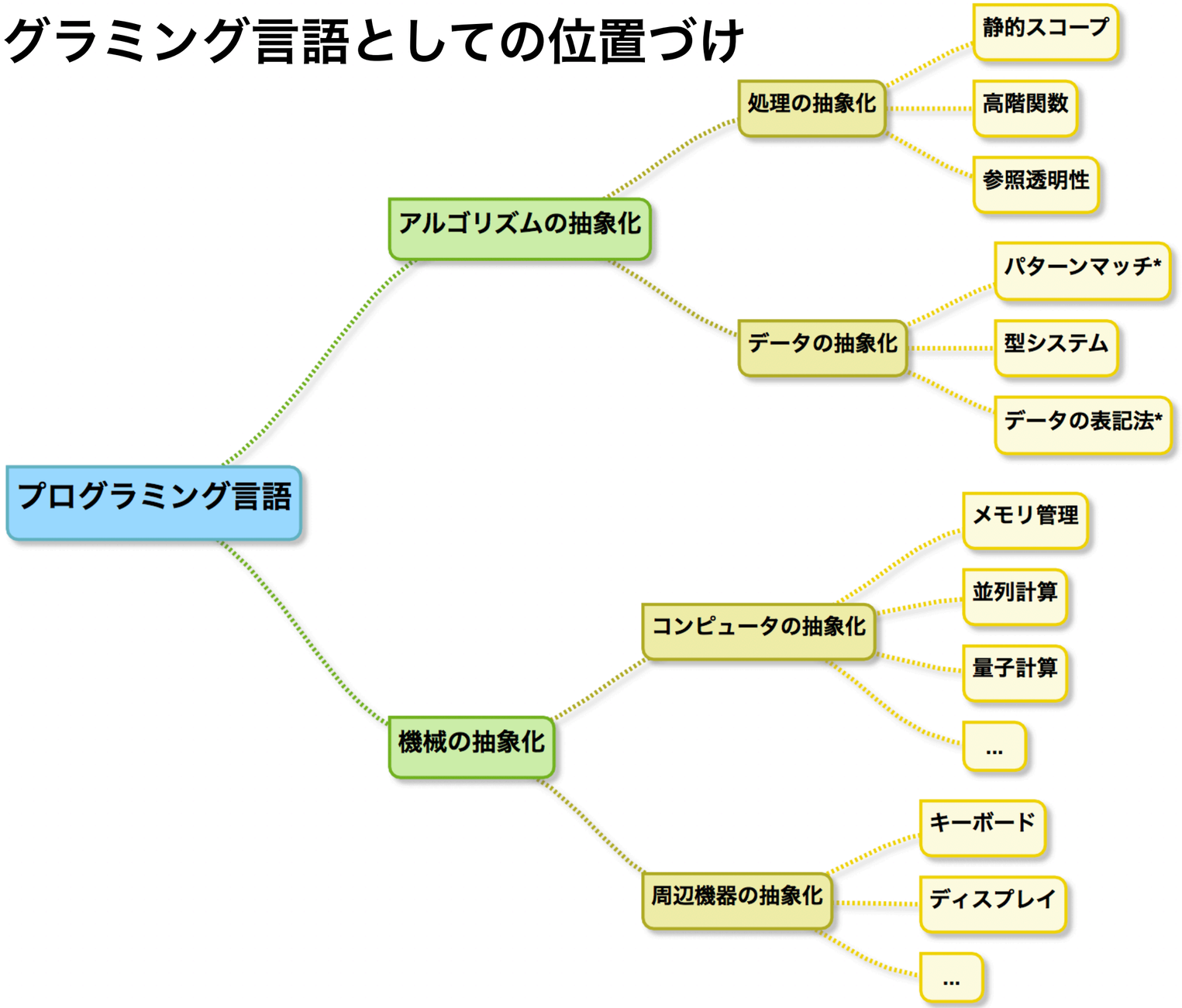
$ cabal install egison

Egison開発の動機

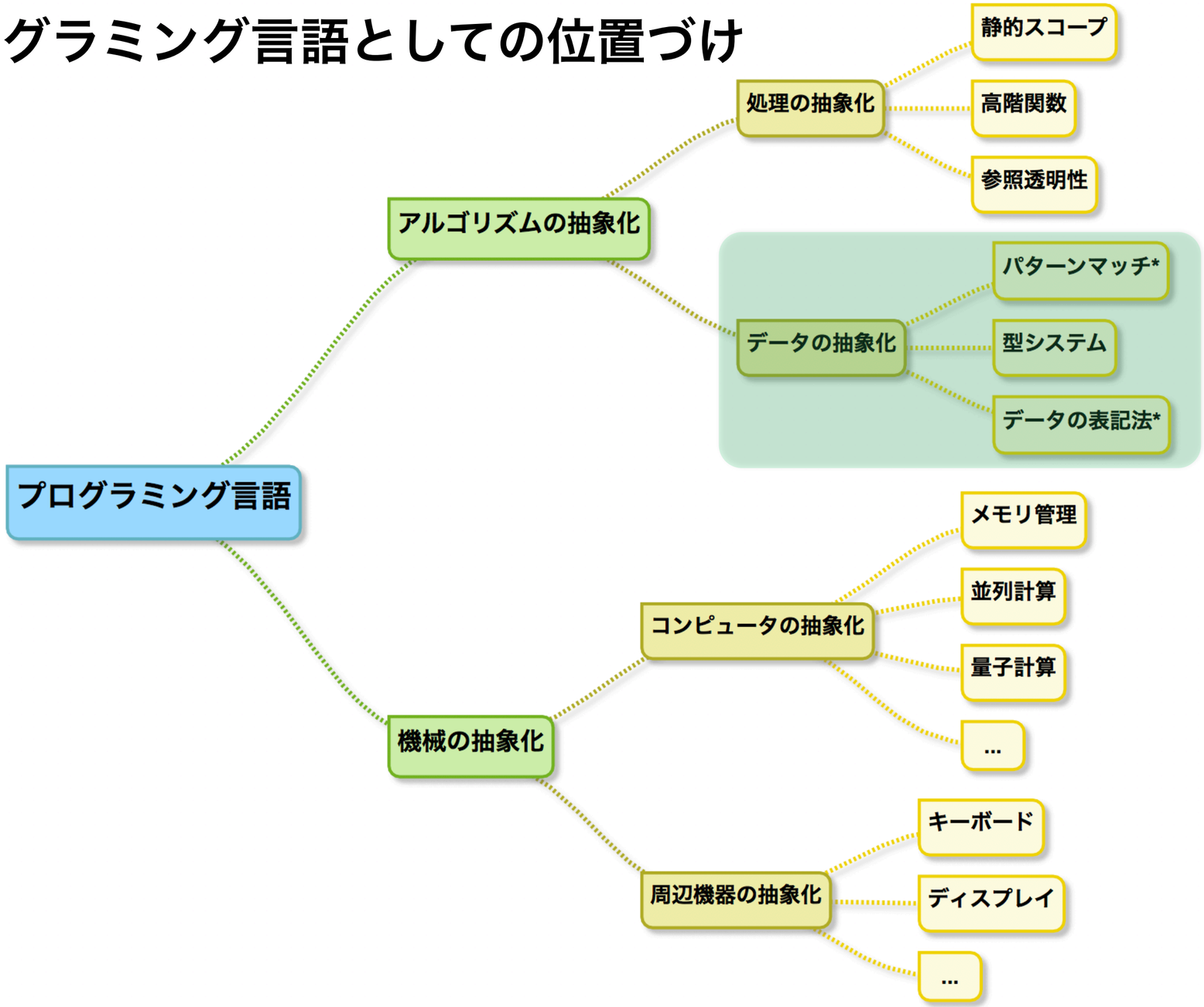
あらゆるアルゴリズムを簡潔に表現したい.

現在は，特に，数理物理の計算を簡潔に表現するための機能の設計・実装にフォーカスしている.

Egisonのプログラミング言語としての位置づけ



Egisonのプログラミング言語としての位置づけ



Egisonの特徴

関数型プログラミング言語.

遅延評価.

S式.

幅広いデータ型に対するパターンマッチ機能.

そのパターンマッチ機能を応用して実装された数式処理システム.

テンソルの添字記法の導入.

Egison独自の機能

関数型プログラミング言語.

遅延評価.

S式.

幅広いデータ型に対するパターンマッチ機能.

そのパターンマッチ機能を応用して実装された数式処理システム.

テンソルの添字記法の導入.

本日の発表の流れ

Egisonの概要

Egison独自の機能

Egison独自の機能

関数型プログラミング言語.

遅延評価.

S式.

幅広いデータ型に対するパターンマッチ機能.

そのパターンマッチ機能を応用して実装された数式処理システム.

テンソルの添字記法の導入.

Egison独自の機能

関数型プログラミング言語.

遅延評価.

S式.

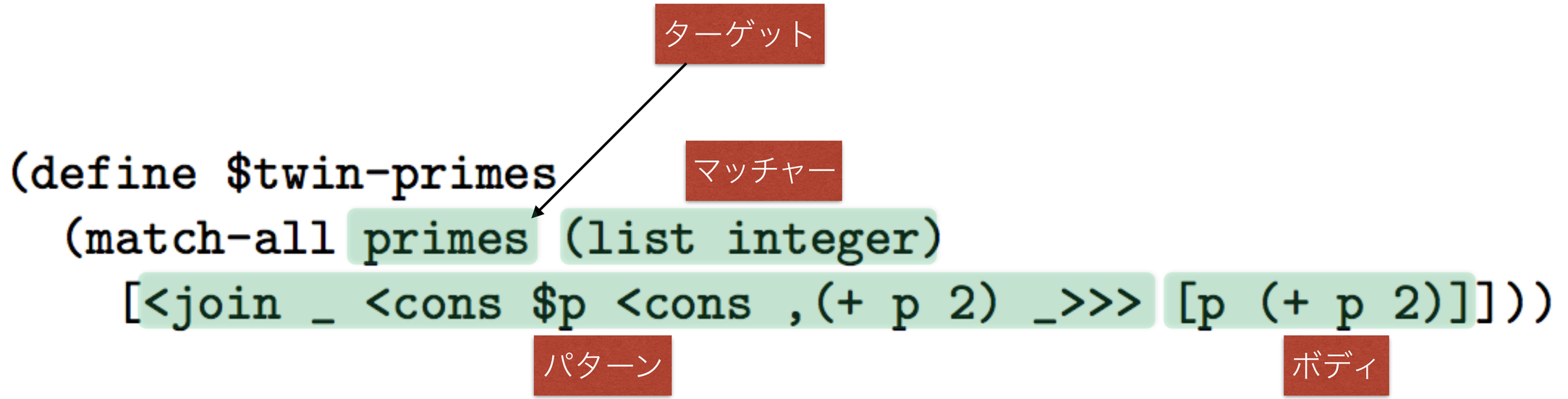
幅広いデータ型に対するパターンマッチ機能.

そのパターンマッチ機能を応用して実装された数式処理システム.

テンソルの添字記法の導入.

match-all式

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




```
(define $twin-primes
  (match-all primes (list integer)
    [<join _ <cons $p <cons ,(+ p 2) _>>> [p (+ p 2)]]))

(take 6 twin-primes)
;{[3 5] [5 7] [11 13] [17 19] [29 31] [41 43]}
```

Egisonのパターンマッチの特徴.

パターンの多相性.

非線形パターンに対する効率的なバックトラッキング.

マッチャーによるパターンマッチの拡張性.

Egisonのパターンマッチの特徴.

パターンの多相性.

非線形パターンに対する効率的なバックトラッキング.

マッチャーによるパターンマッチの拡張性.

パターンの多相性

```
(match-all {1 2 3} (list integer)
  [<cons $x $ts> [x ts]])
; {[1 {2 3}]}

(match-all {1 2 3} (multiset integer)
  [<cons $x $ts> [x ts]])
; {[1 {2 3}] [2 {1 3}] [3 {1 2}]}

(match-all {1 2 3} (set integer)
  [<cons $x $ts> [x ts]])
; {[1 {1 2 3}] [2 {1 2 3}] [3 {1 2 3}]}
```

Egison以前のコレクションに対するパターン

nil パターン：空のコレクションにマッチする.

cons パターン：コレクションを**先頭の要素**と残りのコレクションに分解する.

List	Multiset	Set
`{[1 {2 3}]}`		

Applicable only to Lists.

join パターン：コレクションを**先頭部分のコレクション**と残りのコレクションに分解する.

List	Multiset	Set

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

Applicable only to Lists.

Egisonにより一般化されたコレクションに対するパターン

nil パターン：空のコレクションにマッチする.

cons パターン：コレクションを**ある要素**と残りのコレクションに分解する.

List

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

Multiset

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

Set

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

join パターン：コレクションを**2つのコレクション**に分解する.

List

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

Multiset

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

Set

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

ポーカーの役判定 - 多重集合に対するパターンマッチ

```
(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.

ポーカーの役判定 - ストレートフラッシュ

```
(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>
```

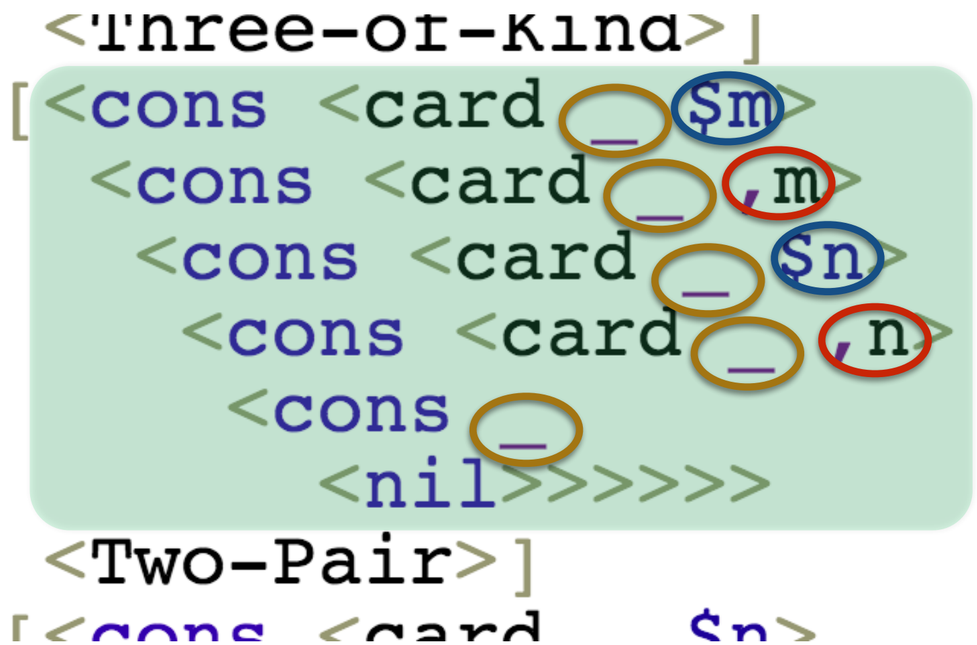
ポーカーの役判定 - ストレートフラッシュ

```
(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>
```

ポーカーの役判定 - 2ペア

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

ポーカーの役判定 - 2ペア



Egisonのパターンマッチの特徴.

パターンの多相性.

非線形パターンに対する効率的なバックトラッキング.

マッチャーによるパターンマッチの拡張性.

非線形パターンに対する効率的なバックトラッキング

非線形パターンとは，同じパターン変数がパターン内に複数あらわれるパターンのことをいう.

```
(match (between 1 n) (multiset integer)
  {[<cons $x <cons ,x _>> "Matched"]
   [_ "Not matched"]})
; Return "Not matched" in O(n^2).

(match (between 1 n) (multiset integer)
  {[<cons $x <cons ,x <cons ,x _>>> "Matched"]
   [_ "Not matched"]})
; Return "Not matched" in O(n^2).

(match (between 1 n) (multiset integer)
  {[<cons $x <cons ,x <cons ,x <cons ,x _>>>> "Matched"]
   [_ "Not matched"]})
; Return "Not matched" in O(n^2).
```

Egisonのパターンマッチの特徴.

パターンの多相性.

非線形パターンに対する効率的なバックトラッキング.

マッチャーによるパターンマッチの拡張性.

多重集合に対するマッチャーの定義

```
(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}]}]
       })))
```

多重集合に対するマッチャー - Consパターン

```
(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}]}]
       })))
```

多重集合に対するマッチャー - Consパターン

```
(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}]}

多重集合に対するマッチャー - Consパターン

```
(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}]}`

多重集合に対するマッチャー - Consパターン

```
(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}]}

多重集合に対するマッチャー - Consパターン

```
(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}]}

パターンマッチのアルゴリズム

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

マッチングステートの簡約の流れ

```
(match-all {2 8 2} (multiset integer) [<cons $m <cons ,m _>> m])
; {2 2}
```

1	MState {[<cons $m <cons ,m _>> (multiset integer) {2 8 2}]} env {}
2	MState {[$m integer 2] [<cons ,m _> (multiset integer) {8 2}]} env {} MState {[$m integer 8] [<cons ,m _> (multiset integer) {2 2}]} env {} MState {[$m integer 2] [<cons ,m _> (multiset integer) {2 8}]} env {}
3	MState {[$m something 2] [<cons ,m _> (multiset integer) {8 2}]} env {}
4	MState {[<cons ,m _> (multiset integer) {8 2}]} env {[m 2]}
5	MState {[,m integer 8] [_ (multiset integer) {2}]} env {[m 2]} MState {[,m integer 2] [_ (multiset integer) {8}]} env {[m 2]}
6	MState {[_ (multiset integer) {8}]} env {[m 2]}
7	MState {[_ something {8}]} env {[m 2]}
8	MState {} env {[m 2]}

Fig. 1. Reduction path of matching states

https://arxiv.org/abs/1808.10603

無限の大きさの探索空間をもつパターンマッチの探索木の整形

```
(take 8 (match-all nats (set integer) [<cons $m <cons $n _>> [m n]]))
; {[1 1] [1 2] [2 1] [1 3] [2 2] [3 1] [1 4] [2 3]}
```

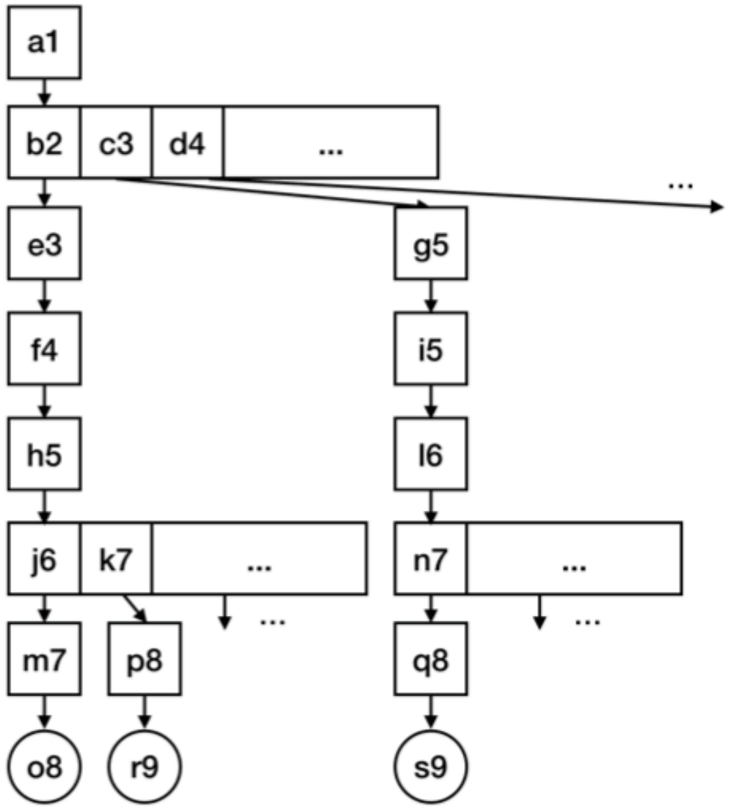


Fig. 2. Search tree

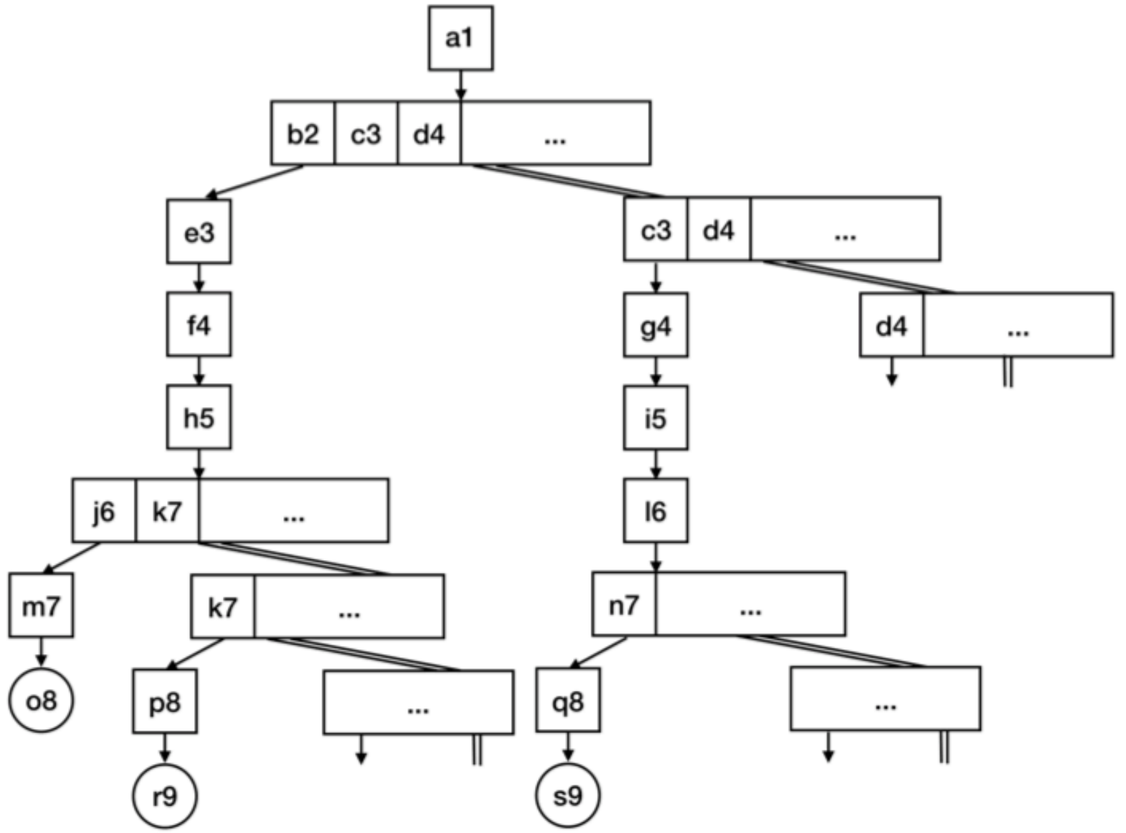


Fig. 3. Binary reduction tree

https://arxiv.org/abs/1808.10603

Egisonのパターンマッチについての論文（APLAS 2018）

Non-linear Pattern Matching with Backtracking for Non-free Data Types

Satoshi Egi[1] and Yuichi Nishiwaki[2]

[1] Rakuten Institute of Technology, Japan
[2] University of Tokyo, Japan

Abstract. *Non-free data types* are data types whose data have no canonical forms. For example, multisets are non-free data types because the multiset $\{a, b, b\}$ has two other equivalent but literally different forms $\{b, a, b\}$ and $\{b, b, a\}$. *Pattern matching* is known to provide a handy tool set to treat such data types. Although many studies on pattern matching and implementations for practical programming languages have been proposed so far, we observe that none of these studies satisfy all the *criteria of practical pattern matching*, which are as follows: i) efficiency of the backtracking algorithm for non-linear patterns, ii) extensibility of matching process, and iii) polymorphism in patterns.
This paper aims to design a new *pattern-matching-oriented* programming language that satisfies all the above three criteria. The proposed language features clean Scheme-like syntax and efficient and extensible pattern matching semantics. This programming language is especially useful for the processing of complex non-free data types that not only include multisets and sets but also graphs and symbolic mathematical expressions. We discuss the importance of our criteria of practical pattern matching and how our language design naturally arises from the criteria. The proposed language has been already implemented and open-sourced as the Egison programming language.

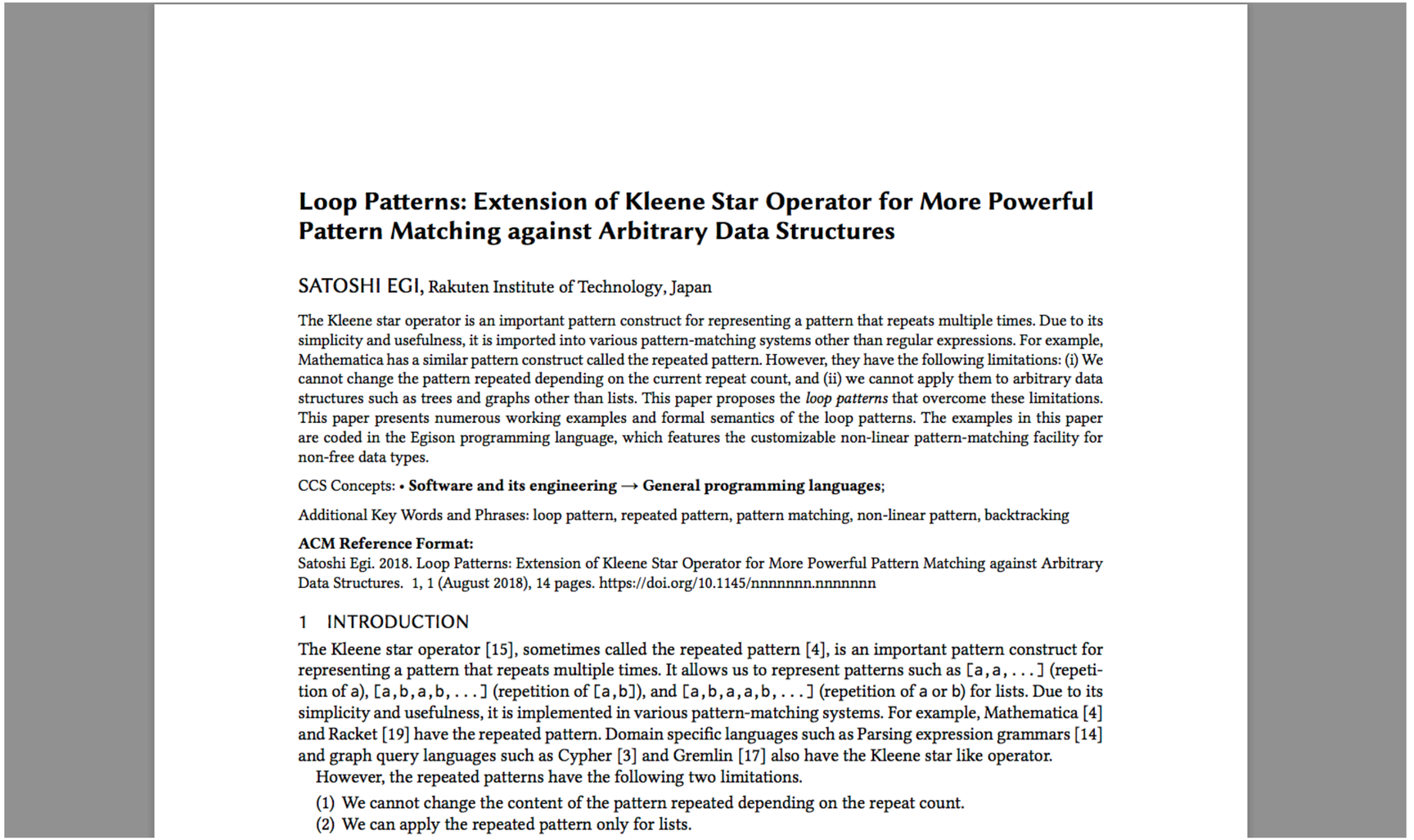
1 Introduction

Pattern matching is an important feature of programming languages featuring data abstraction mechanisms. Data abstraction serves users with a simple method for handling data structures that contain plenty of complex information. Using pattern matching, programs using data abstraction become concise, human-readable, and maintainable. Most of the recent practical programming

https://arxiv.org/abs/1808.10603

ループパターンについての論文（Scheme Workshop 2018）

ツリーやグラフのパターンマッチを含む色々な実例が紹介されている.



Loop Patterns: Extension of Kleene Star Operator for More Powerful Pattern Matching against Arbitrary Data Structures

SATOSHI EGI, Rakuten Institute of Technology, Japan

The Kleene star operator is an important pattern construct for representing a pattern that repeats multiple times. Due to its simplicity and usefulness, it is imported into various pattern-matching systems other than regular expressions. For example, Mathematica has a similar pattern construct called the repeated pattern. However, they have the following limitations: (i) We cannot change the pattern repeated depending on the current repeat count, and (ii) we cannot apply them to arbitrary data structures such as trees and graphs other than lists. This paper proposes the *loop patterns* that overcome these limitations. This paper presents numerous working examples and formal semantics of the loop patterns. The examples in this paper are coded in the Egison programming language, which features the customizable non-linear pattern-matching facility for non-free data types.

CCS Concepts: • **Software and its engineering → General programming languages**;

Additional Key Words and Phrases: loop pattern, repeated pattern, pattern matching, non-linear pattern, backtracking

ACM Reference Format:
Satoshi Egi. 2018. Loop Patterns: Extension of Kleene Star Operator for More Powerful Pattern Matching against Arbitrary Data Structures. 1, 1 (August 2018), 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Kleene star operator [15], sometimes called the repeated pattern [4], is an important pattern construct for representing a pattern that repeats multiple times. It allows us to represent patterns such as `[a,a,...]` (repetition of `a`), `[a,b,a,b,...]` (repetition of `[a,b]`), and `[a,b,a,a,b,...]` (repetition of `a` or `b`) for lists. Due to its simplicity and usefulness, it is implemented in various pattern-matching systems. For example, Mathematica [4] and Racket [19] have the repeated pattern. Domain specific languages such as Parsing expression grammars [14] and graph query languages such as Cypher [3] and Gremlin [17] also have the Kleene star like operator.

However, the repeated patterns have the following two limitations.

(1) We cannot change the content of the pattern repeated depending on the repeat count.
(2) We can apply the repeated pattern only for lists.

https://arxiv.org/pdf/1809.03252.pdf

パターンマッチに関係するEgisonライブラリ

egison/lib/core/

lib/core/collection.egi: コレクションに対するマッチャーが定義されている.

Egison独自の機能

関数型プログラミング言語.

遅延評価.

S式.

幅広いデータ型に対するパターンマッチ機能.

そのパターンマッチ機能を応用して実装された数式処理システム.

テンソルの添字記法の導入.

シンボリックな計算

```
(* (+ 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))
```

数式の簡約 - 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
```

1の7乗根

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

```
(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)
```

微分演算子の定義

```
(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)))]
       })))
```

微分演算子の適用

```
(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)
```

テイラー展開の定義

```
(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)))))))
```

テイラー展開の適用

```
(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)}
```

シンボリックな計算に関係するEgisonライブラリ

egison/lib/math/

lib/math/expression.egi: 数式に対するマッチャーが定義されている.

lib/math/normalize.egi: 数式の簡約規則が定義されている.

lib/math/algebra: 代数方程式の解や行列式の計算が定義されている.

lib/math/analysis: 微分演算子やテイラー展開が定義されている.

Egison独自の機能

関数型プログラミング言語.

遅延評価.

S式.

幅広いデータ型に対するパターンマッチ機能.

そのパターンマッチ機能を応用して実装された数式処理システム.

テンソルの添字記法の導入.

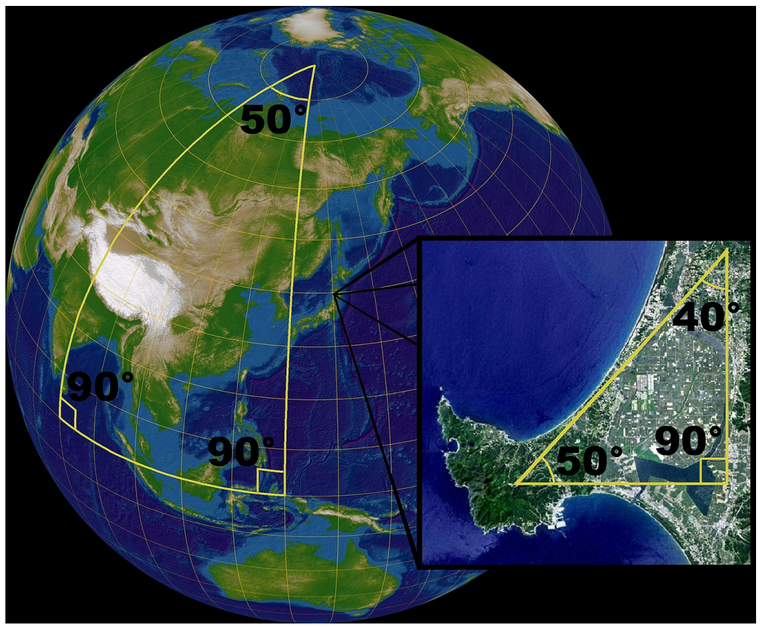
リーマン幾何学とは？

リーマン幾何学は，曲がった空間について調べるための理論である.

3次元空間の住民である我々は，2次元の曲面が曲がっていることを当たり前に認識できる.

しかし，3次元の曲がった空間はどうだろうか？

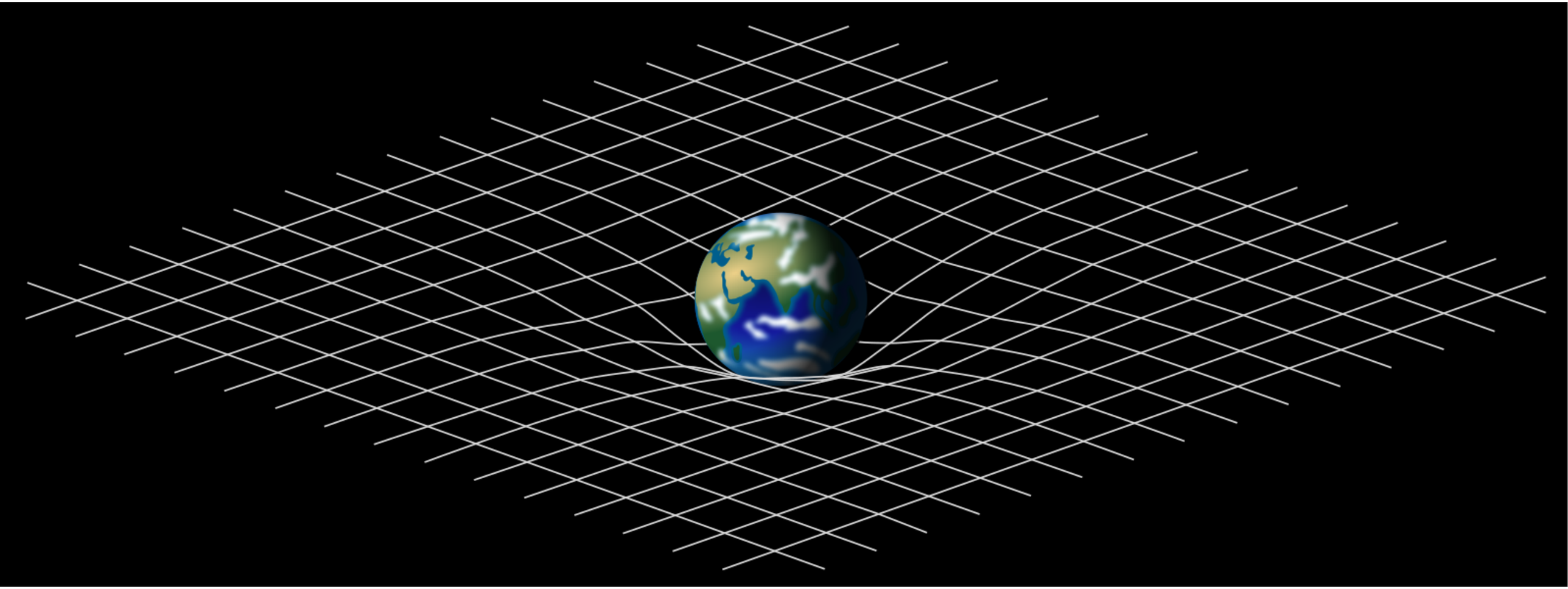
我々の住んでいるこの宇宙は，より高次元を認識できる存在からみると曲がっているのだろうか？



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

アインシュタインの一般相対性理論

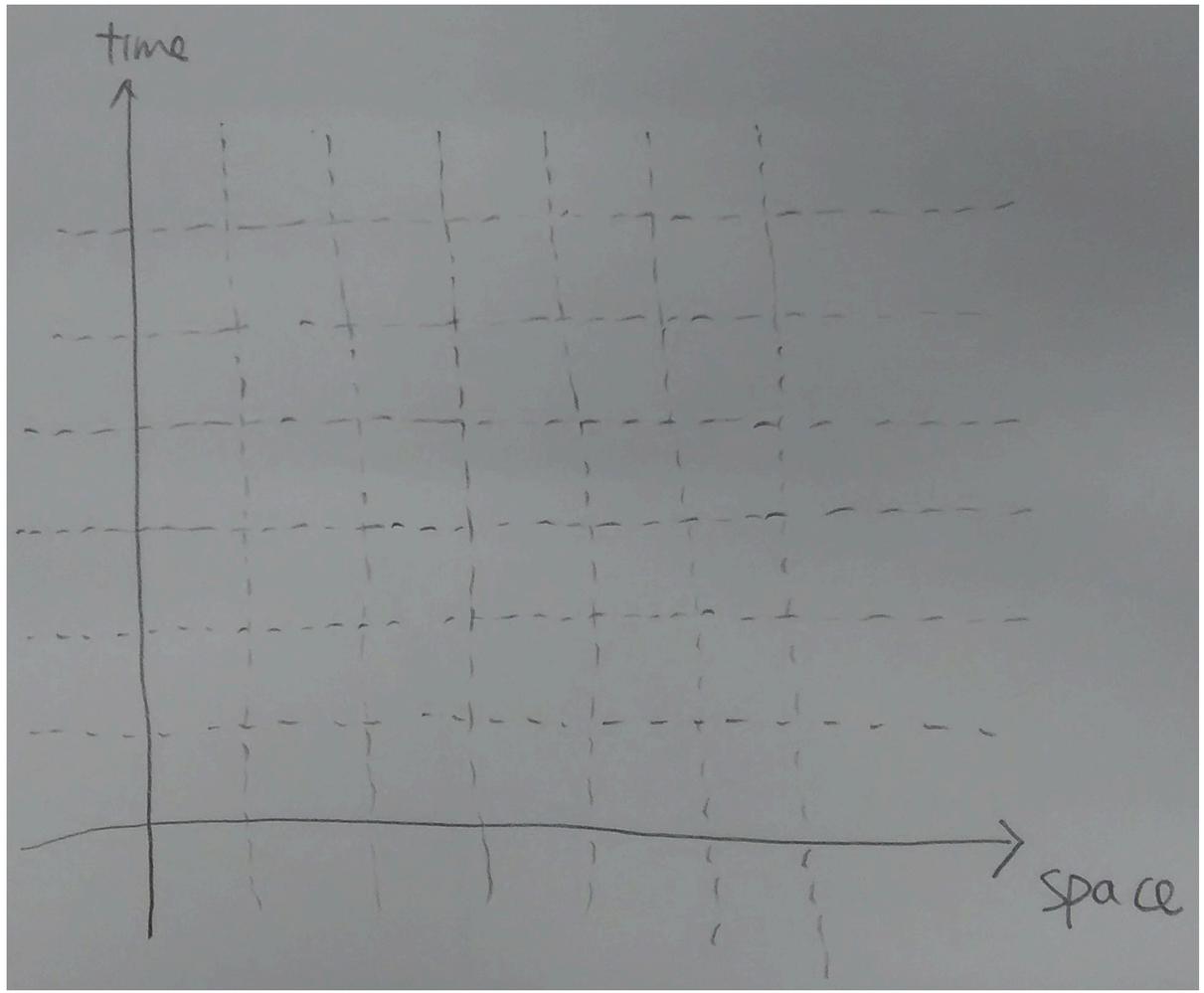
一般相対性理論は，重力を4次元時空の歪みとして説明する.



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

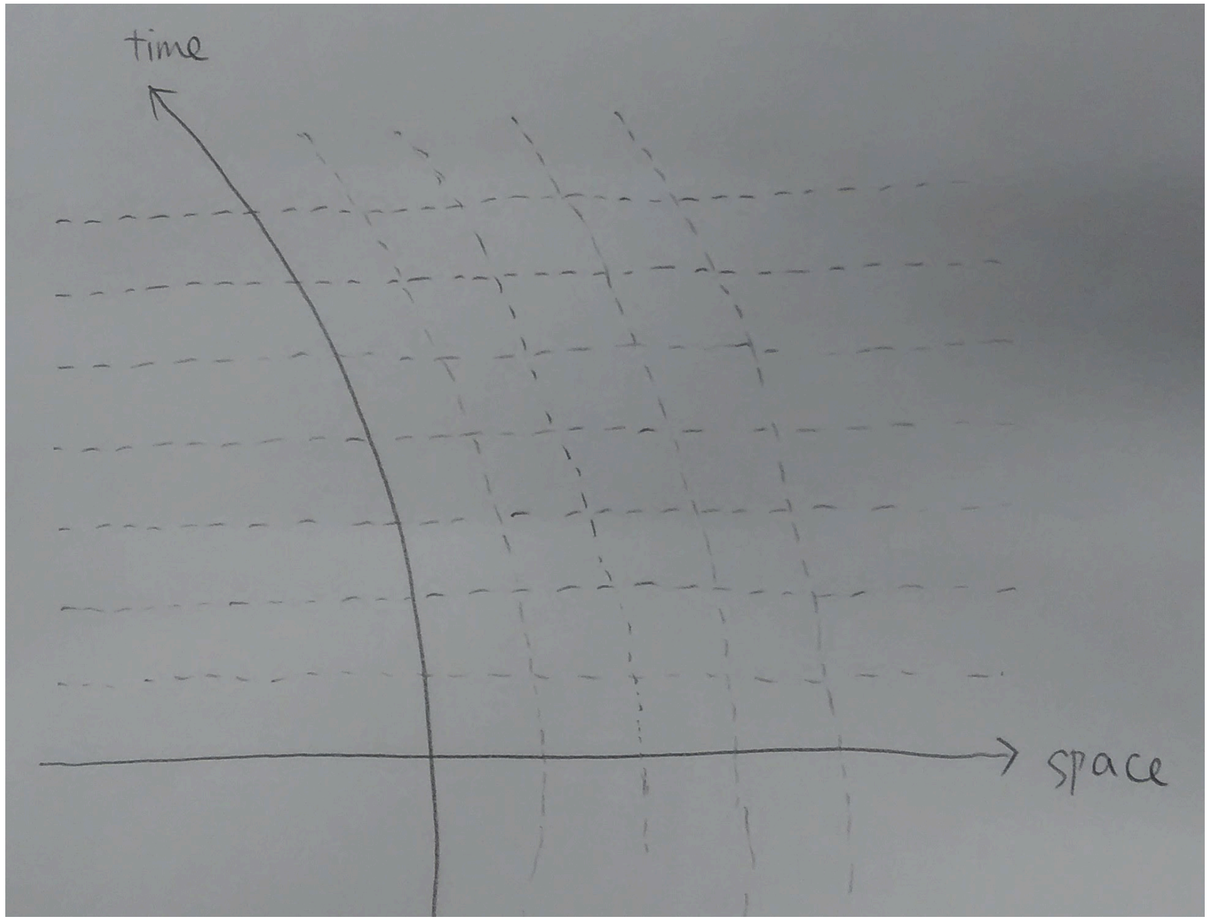
一般相対性理論の直感的な説明

我々の認識では，この時空間はまっすぐである.



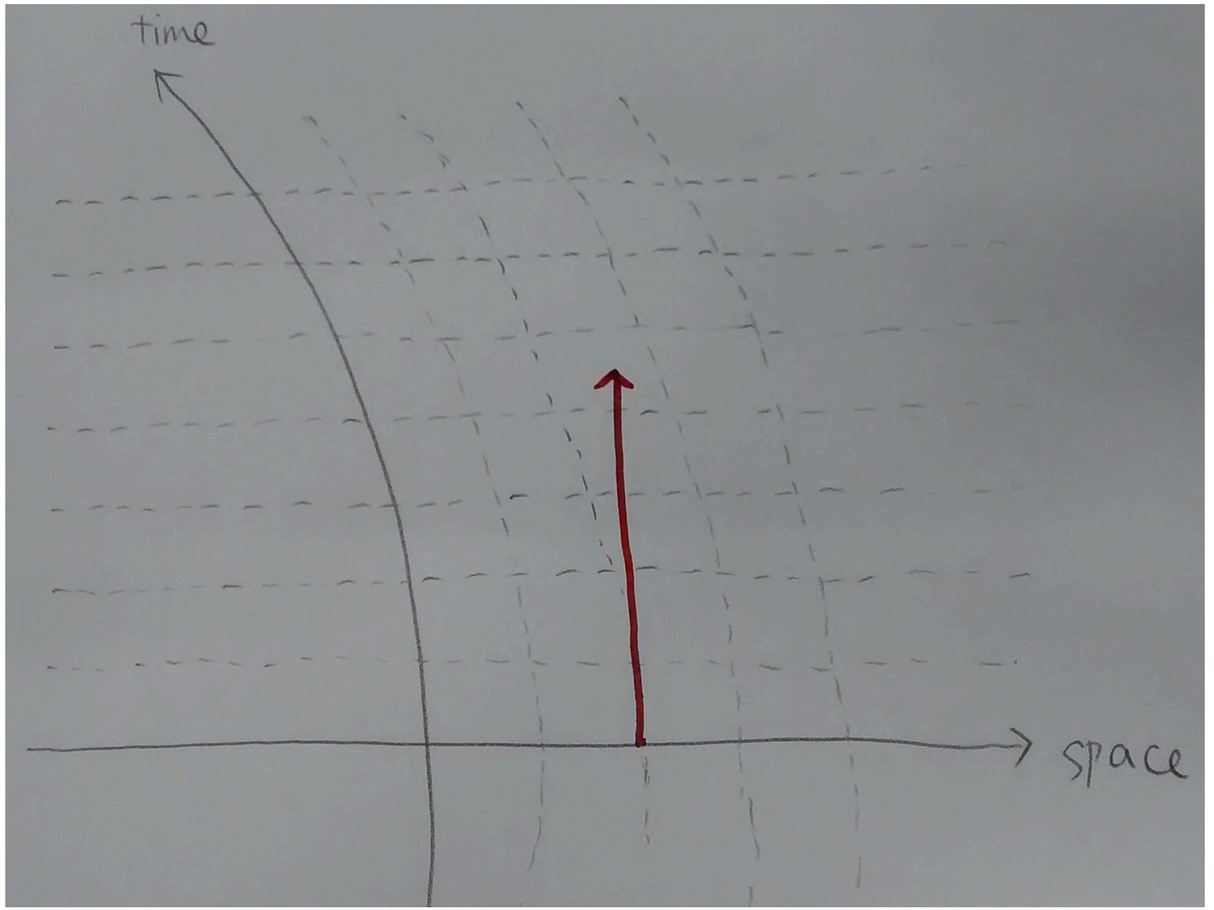
一般相対性理論の直感的な説明

しかし，一般相対性理論によるとこの時空間は曲がっている.



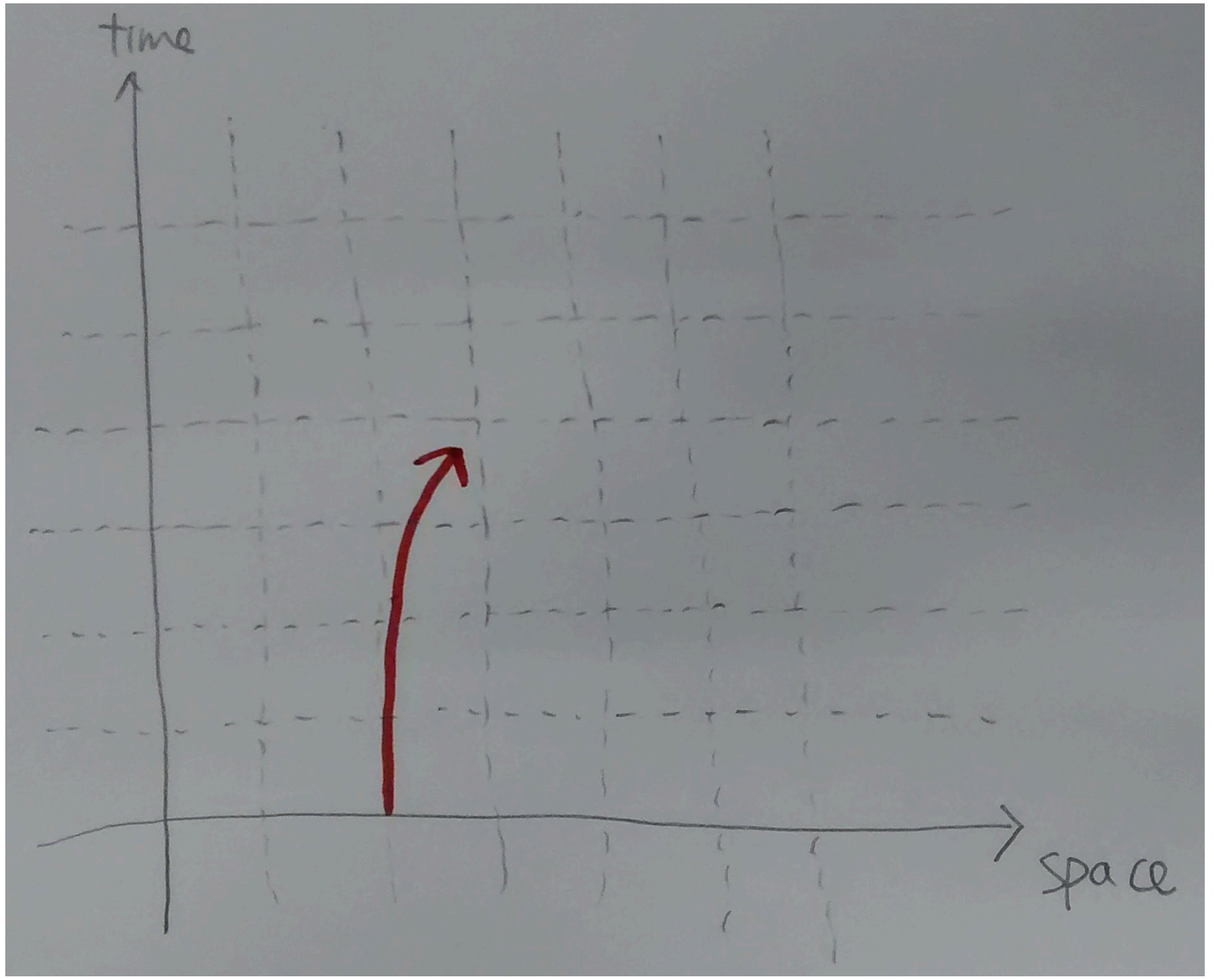
一般相対性理論の直感的な説明

この曲がった時空間を物体はまっすぐ進む.



一般相対性理論の直感的な説明

我々の認識では，その動きが曲がっているように見える.



リーマン幾何学で使われる記法

リーマン幾何学の数式は，**テンソルの添字記法**が使われる.

Egisonはこの記法をそのままプログラミングで使うことをサポートしている.

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

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

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

Egisonにおけるテンソルの添字記法

Egisonでは，"∂/∂"（変微分演算子）や "."（テンソルのかけ算）をはじめとする任意の関数をテンソルに対して添字記法を使って直接適用できる.

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

Formula of Riemann curvature tensor

~: 上添字

_: 下添字

```
(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

Egisonの手法 - テンソルの添字記法のデモ

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

リーマン曲率テンソルの公式

```
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言語によるリーマン曲率テンソルの公式の表現

```
(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によるリーマン曲率テンソルの公式の表現

~: 上添字

_: 下添字

Egisonによる添字記法をプログラミングに導入するための方法

関数を2種類に分類する:

- テンソルの成分ごとに処理が自動でマップされるべき関数.

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

- テンソルをそのまま引数としてとるべき関数.

e.g. テンソル同士のかけ算，行列式の計算，...

この2種類の関数の適用に対して，上手に添字の簡約規則を定義すると，添字記法をプログラミングに導入できる.

Egisonの手法 - テンソルの添字記法の導入 (その1)

仮引数の先頭に"$"が付いている場合，テンソルの成分ごとに処理がマップされる．

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

スカラー仮引数の例としてmin関数の定義

仮引数の先頭に"%"が付いている場合，テンソルがそのまま引数に渡される．

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

テンソル仮引数の例として"."関数の定義

Egisonの手法 - テンソルの添字記法の導入 (その2)

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

スカラー仮引数の例としてmin関数の定義

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

違う添字を持つベクトルへのmin関数の適用

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

同じ添字を持つベクトルへのmin関数の適用

Egisonの手法 - テンソルの添字記法の導入 (その3)

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

テンソル仮引数の例として"."関数の定義

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

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

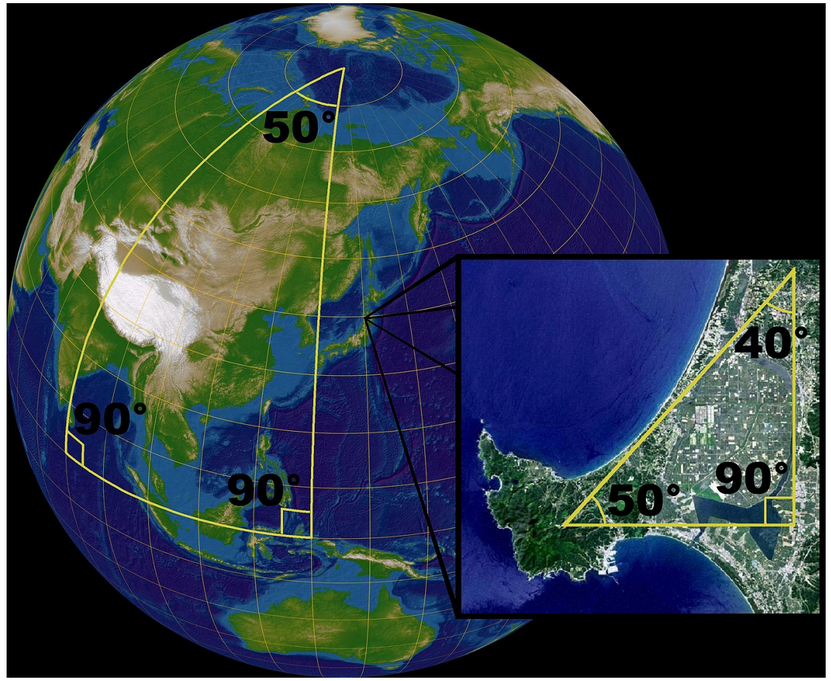
$$\begin{pmatrix}1\\2\\3\end{pmatrix}_{i} \cdot \begin{pmatrix}10\\20\\30\end{pmatrix}_{j} = contract(+, \begin{pmatrix}10 & 20 & 30\\20 & 40 & 60\\30 & 60 & 90\end{pmatrix}_{ij}) = \begin{pmatrix}10 & 20 & 30\\20 & 40 & 60\\30 & 60 & 90\end{pmatrix}_{ij}$$

"."関数のいろいろな添字の組み合わせのベクトルへの適用

応用：リーマン幾何学の計算

球面のリーマン曲率テンソルの計算

```
;; 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

局所座標 - 球面

```
;;;
;;; 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

リーマン計量 - 球面

```
;;
;; 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)) |] |]~#~#
```

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

第一種クリストッフェル記号 - 球面

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

```
;;
;; 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

第二種クリストッフェル記号 - 球面

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

```
;;
;; 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

リーマン曲率テンソル - 球面

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

```
;;
;; 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

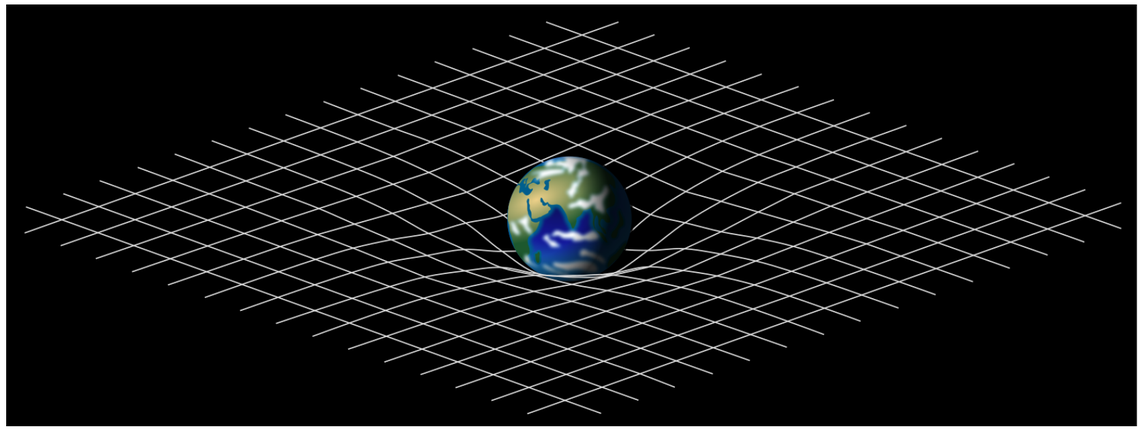
リッチ曲率とスカラー曲率 - 球面

```
;;
;; 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

Schwarzschild空間のリーマン曲率テンソルの計算

```
;; 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

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

添字記法をプログラミングへ導入する手法についての論文（Scheme Workshop 2017）

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 (微分形式)

微分形式とは (その1)

微分形式を使うと座標系に依存しない微分方程式を記述できる.

通常の偏微分方程式は，選択した座標系によって微分方程式の表現が変わってしまう.

- 2次元デカルト座標のラプラシアンの偏微分方程式

- 2次元極座標のラプラシアンの偏微分方程式

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

微分形式とは (その2)

ウェッジ積(∧)・外微分(d)・内部積(ı)・ホッジ作用素(*)という限られたオペレーターを組み合わせて，任意の座標系で表現を変えずに成り立つ微分方程式を記述する.

これらの作用素は，grad・rot・divの一般化でもある.

grad = d

rot = d

$\delta = (-1)^{p} * {}^{-1} d *$

div = δ

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

座標系の計量を与えると，通常の偏微分方程式を得ることができる.

ホッジ作用素のみ定義が座標系の計量に依存している.

微分形式の記法のFormuraへの導入の動機

物理シミュレーションに使われる微分方程式の多くは，デカルト座標系以外の座標系では，通常の偏微分方程式による記述が複雑になることが多い.

しかし，微分形式の言葉で記述した微分方程式を実行できるようになれば，簡潔なプログラムで，これらの物理シミュレーションをできるようになる.

Egisonによるホッジラプラシアンの定義とデモ

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

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

```
(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)))]})))

(define $f (function [r θ]))
(Δ f)
;(/ (+ (* -1 f|θ|θ) (* -1 r f|r) (* -1 r^2 f|r|r)) r^2)
```

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

Egisonによるナビエ・ストークス方程式の展開(デカルト座標系)

In [22]:

```
(with-symbols {i j}
  (+ (∂/∂ u t)
     (Lie u u)
     (* (/ -1 2) (d (. g~i~j u_i u_j)))
     (* (/ 1 ϱ) (- (d p) (* η (Δ u))))))
```

物理学会誌2017年1月号 散逸系の変分原理 深川宏樹

$$\begin{pmatrix} \frac{\frac{\partial u_1}{\partial t}\rho+u_2\frac{\partial u_1}{\partial y}\rho+u_3\frac{\partial u_1}{\partial z}\rho+u_1\frac{\partial u_1}{\partial x}\rho+\frac{\partial p}{\partial x}-\eta\frac{\partial^2 u_1}{\partial^2 x}-\eta\frac{\partial^2 u_1}{\partial^2 z}-\eta\frac{\partial^2 u_1}{\partial^2 y}}{\rho} \\ \frac{\frac{\partial u_2}{\partial t}\rho+u_1\frac{\partial u_2}{\partial x}\rho+u_3\frac{\partial u_2}{\partial z}\rho+u_2\frac{\partial u_2}{\partial y}\rho+\frac{\partial p}{\partial y}-\eta\frac{\partial^2 u_2}{\partial^2 y}-\eta\frac{\partial^2 u_2}{\partial^2 z}-\eta\frac{\partial^2 u_2}{\partial^2 x}}{\rho} \\ \frac{\frac{\partial u_3}{\partial t}\rho+u_1\frac{\partial u_3}{\partial x}\rho+u_2\frac{\partial u_3}{\partial y}\rho+u_3\frac{\partial u_3}{\partial z}\rho+\frac{\partial p}{\partial z}-\eta\frac{\partial^2 u_3}{\partial^2 z}-\eta\frac{\partial^2 u_3}{\partial^2 y}-\eta\frac{\partial^2 u_3}{\partial^2 x}}{\rho} \end{pmatrix}$$

微分形式の記法をプログラミング言語に導入する関連研究

Karczmarczuk, J. : Functional Coding of Differential Forms (1999)
Sussman, G. J. and Wisdom, J. : Functional Differential Geometry (MIT Press, 2013)

Egisonの利点

- より簡潔な記述で，微分形式のための演算子を定義する仕組みを提供している.
- テンソルの添字記法も同時にサポートしており，テンソルを値にとる微分形式も扱える.

関連研究との比較 - 外微分の定義

```
(define (exterior-derivative-procedure kform)
  (let ((k (get-rank kform)))
    (if (fix:= k 0)
    (differential-of-function kform)
    (let ((the-k+1form
           (lambda vectors
         (assert (fix:= (length vectors) (fix:+ k 1)))
         (lambda (point)
           (let ((n (s:dimension point)))
             (if (fix:< k n)
             (sigma
              (lambda (i)
                (let ((rest (delete-nth i vectors)))
                  (+ (* (if (even? i) +1 -1)
                    (((ref vectors i) (apply kform rest))
                     point))
                   (sigma
                    (lambda (j)
                      (* (if (even? (fix:+ i j)) +1 -1)
                         ((apply kform
                             (cons
                          (commutator (ref vectors i)
                                  (ref vectors j))
                          ;; j-1 because already deleted i.
                          (delete-nth (fix:- j 1)
                                  rest)))
                         point)))
                    (fix:+ i 1) k))))
              0 k)
             0))))))
      (procedure->nform-field the-k+1form
                  `(d ,(diffop-name kform))
                  (fix:+ (get-rank kform) 1))))))

(define exterior-derivative
  (make-operator exterior-derivative-procedure
         'd
         'exterior-derivative))

(define d exterior-derivative)
```

Sussman, G. J. and Wisdom, J. : Functional Differential Geometry (MIT Press, 2013)

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

Egison

Egisonの手法 - 微分形式の記法の導入

省略されたテンソルの添字についての補完のルールを適切に設定すれば，テンソルの添字記法を導入した手法により，そのまま微分形式の記法も導入できる.

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

リーマン曲率テンソルの公式

曲率形式の公式

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

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

デフォルトで同じ添字の組み合わせを補完する.

"!"が関数適用の前に付加されていたら，別々の添字の組み合わせを補完する.

Egisonによるウェッジ積の定義とデモ

```
(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

Egisonによる外微分の定義とデモ

```
(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)))

(define $f (function [x y z]))

(d f)
;[| f|x f|y f|z |]

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

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

Egisonによる内部積の定義とそれを用いて定義されたリー微分

```
(define $ι
  (lambda [%X %Y]
    (with-symbols {i}
      (* (df-order Y) (. X...~i (df-normalize Y)..._i)))))
```

```
(define $Lie
  (lambda [%X %Y]
    (match (df-order Y) integer
      {[,0 (ι X (d Y))]
       [,N (d (ι X Y))]
       [_ (+ (ι X (d Y)) (d (ι X Y)))]})))
```

Egisonによるホッジ作用素の定義とデモ (ユークリッド空間)

```
(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

Egisonによるホッジ作用素の定義とデモ (ミンコフスキー空間)

```
(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)
```

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

Egison数学ノート

整数論

初等整数論	トリボナッチ数列 オイラーのϕ関数
二次体	$\mathbb{Z}[i]$の素数 $\mathbb{Z}[w]$の素数

代数学

代数方程式	二次方程式 三次方程式 四次方程式
1のn乗根	1の5乗根 1の7乗根 1の9乗根 1の17乗根

解析学

微分	2次元極座標のラプラシアン 3次元極座標のラプラシアン
無限級数	オイラーの公式
フーリエ解析	ライプニッツの公式

幾何学

微分幾何	曲面のガウス曲率 S^2のリーマン曲率テンソル S^3のリーマン曲率テンソル S^4のリーマン曲率テンソル S^5のリーマン曲率テンソル S^7のリーマン曲率テンソル T^2のリーマン曲率テンソル $S^2 \times S^3$のリーマン曲率テンソル シュワルツシルト計量 フリードマン・ルメートル・ロバートソン・ウォーカー計量
微分形式	ウェッジ積 外微分 E^3のホッジ作用素 ミンコフスキー空間のホッジ作用素 極座標のホッジラプラシアン 球座標のホッジラプラシアン 曲率形式
物理への応用	ベクトル解析 $U(1)$ゲージ理論のヤンミルズ方程式
特性類	S^2のオイラー形式 T^2オイラー形式 CP_1の直線バンドルのチャーン形式

https://www.egison.org/math

Egisonについての論文

Pattern-Matching Papers:

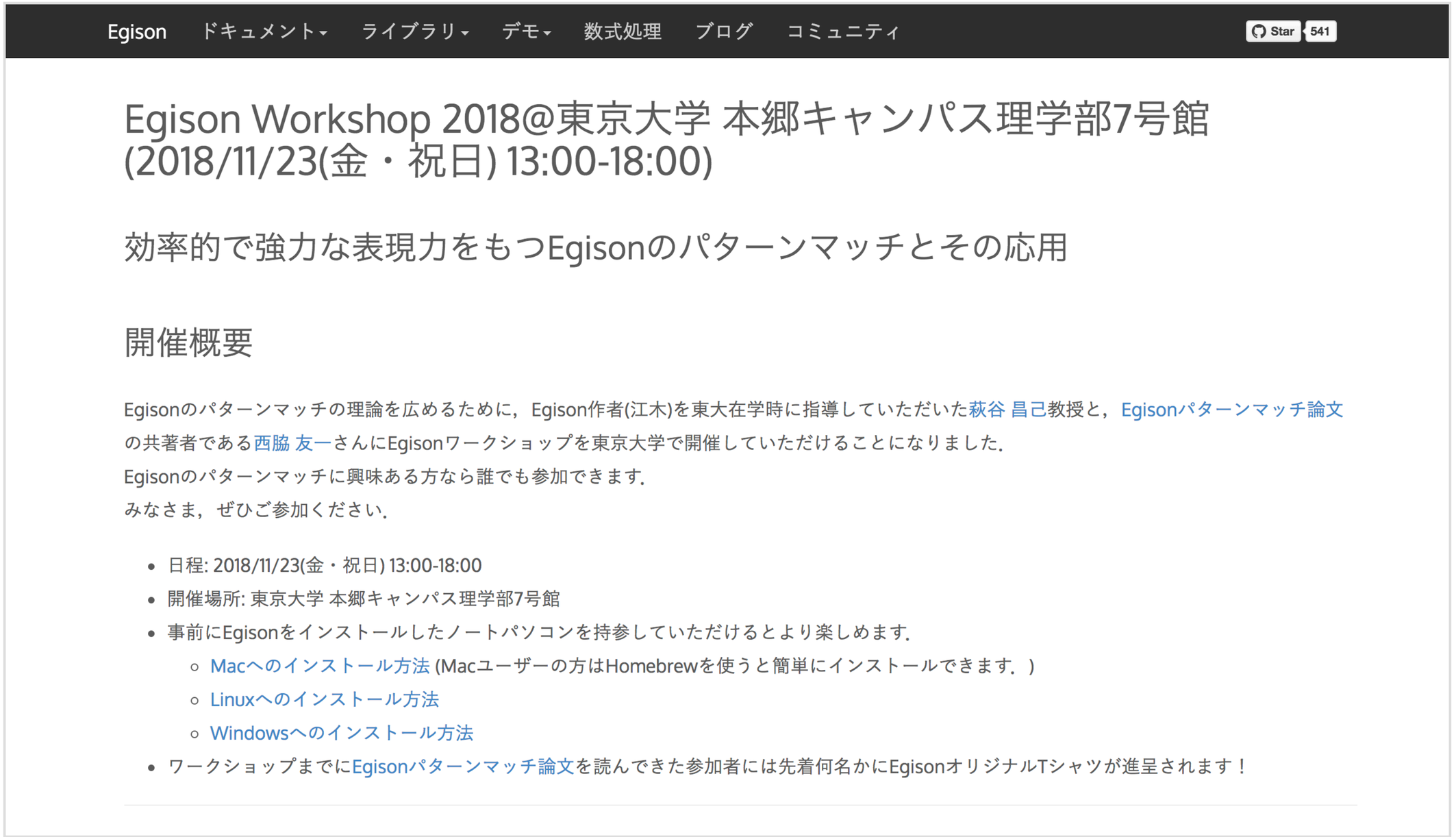
S. Egi, Y. Nishiwaki: Non-linear Pattern Matching with Backtracking for Non-free Data Types (APLAS 2018) https://arxiv.org/pdf/1808.10603.pdf

S. Egi: Loop Patterns: Extension of Kleene Star Operator for More Expressive Pattern Matching against Arbitrary Data Structures (Scheme Workshop 2018) https://arxiv.org/pdf/1809.03252.pdf

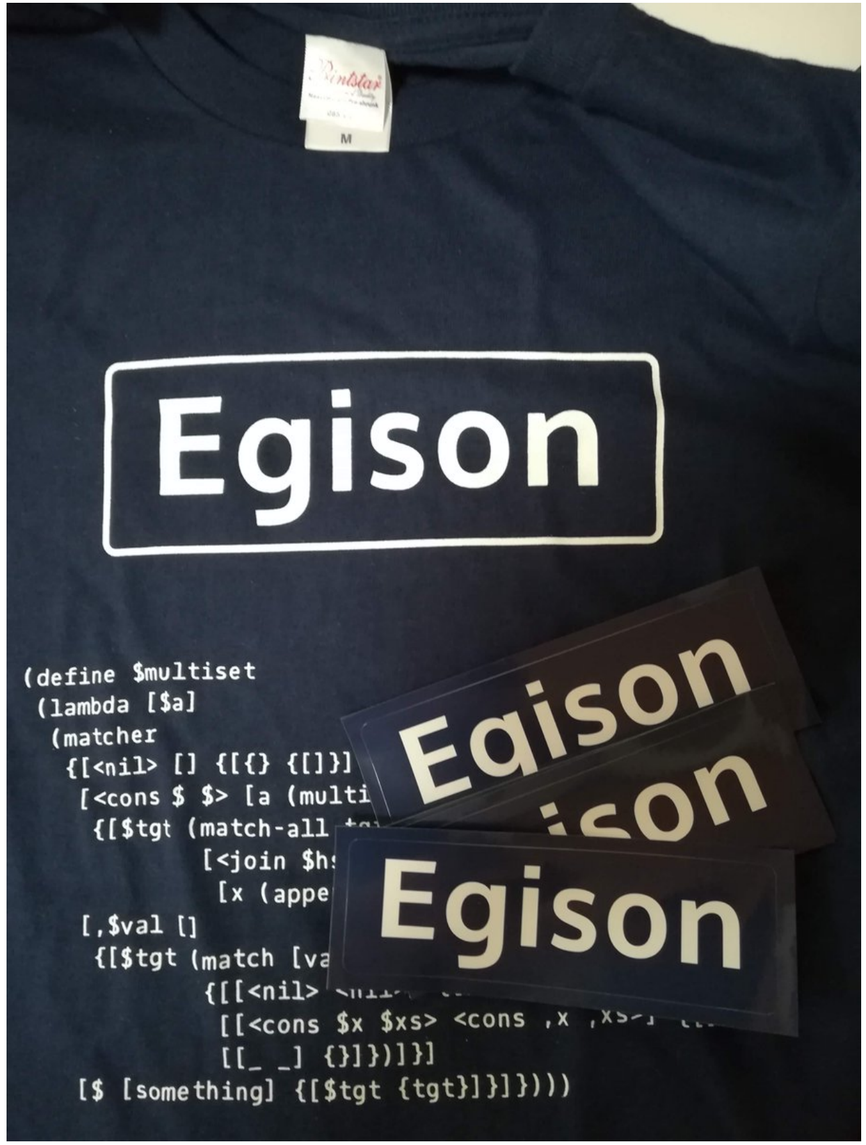
Tensor Papers:

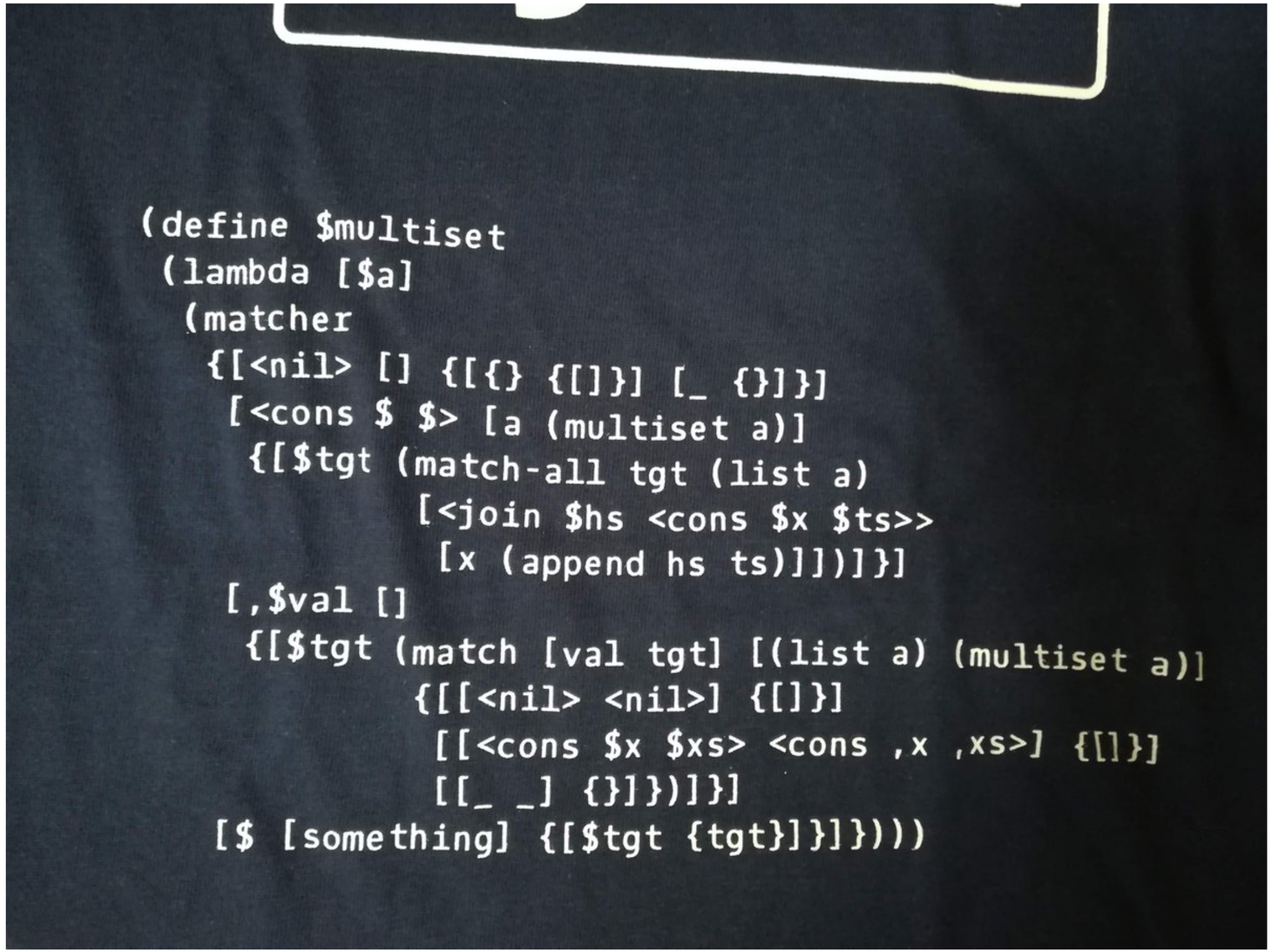
S. Egi: Scalar and Tensor Parameters for Importing Tensor Index Notation including Einstein Summation Notation (Scheme Workshop 2017) https://arxiv.org/pdf/1702.06343.pdf

Egison Workshop 2018



Egisonグッズ


Egison
Egison
Egison
(define $multiset
(lambda [$a]
(matcher
{[<nil> [] {[{} {[]}]
[<cons $ $> [a (multi
{[$tgt (match-all
[<join $hs
[x (appe
[,$val []
{[$tgt (match [va
{[[<nil>
[[<cons $x $xs> <cons ,x
[[_ _] {}]})]]]
[$ [something] {[$tgt {tgt}]}]})))


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

Egison開発アルバイト募集

- インタプリタの改良.

- 数式処理システムの改良.

- 型システム実装の続き.

- Formura（有限差分法による物理シミュレーションプログラムのコンパイラ）と連携するためのプログラム開発.

- マニュアル執筆.


Rakuten