# Egisonの型システムとその型推論器の実装

河田 旺(@a_kawashiro)

# Egisonに型をつけるインターン

- 型システムの設計

  Egisonの構文の定式化と型付け規則の設計

- 型推論器の実装

  Hindley-Milner型推論アルゴリズムを使った型推論器の実装

# Egisonに型をつけるインターン

- 型システムの設計

  Egisonの構文の定式化と型付け規則の設計

- 型推論器の実装

  Hindley-Milner型推論アルゴリズムを使った型推論器の実装

# 型とは何か

プログラム(項)を分類する種類

Int

```
10

10 + 10
```

Int -> Int

```
(\x -> x)

(+ 10)
```

String

```
"Hello"

"W" ++ "orld"
```

……

# 型の利点

- エラーの事前検出

```
"The result is " ++ n
```

```
"The result is " ++ show(n)
```

➡ プログラムを実行する前に間違っている部分が分かる

- ドキュメントとしての利用

```
maybe :: b -> (a -> b) -> Maybe a -> b
```

➡ プログラムの挙動が型から読み取れる

# Haskellでの型の例

```
Prelude> :t 10
10 :: Num t => t
```

```
Prelude> :t "Hello"
"Hello" :: [Char]
```

```
Prelude> :t (\x -> x + 10)
(\x -> x + 10) :: Num a => a -> a
```

# Egisonでの型の例

```
> (print-type-of 10)
10 :: Integer
```

```
> (print-type-of "Hello")
"Hello" :: String
```

```
> (print-type-of (lambda [$x] (b.+ x 10)))
(lambda [$x] (b.+ x 10)) :: (Integer -> Integer)
```

# 型の付け方

## 型付け規則を組み合わせる

**Egisonの型付け規則の一部**

$$\frac{\Gamma \vdash M_1 : \texttt{Bool} \qquad \Gamma \vdash M_2 : T \qquad \Gamma \vdash M_3 : T}{\Gamma \vdash \texttt{(if}\ M_1\ M_2\ M_3\texttt{)} : T}\textsc{T-If}$$

$$\frac{\Gamma \mathbin{+\!\!+} \{x_1 : S_1, x_2 : S_2, ...x_n : S_n\} \vdash M : T}{\Gamma \vdash \texttt{(lambda [\$}x_1\ \texttt{\$}x_2\ ...\ \texttt{\$}x_n\texttt{]}\ M\texttt{)} : \texttt{[}S_1, S_2, ...\texttt{]} \to T}\textsc{T-Abs}$$

$$\frac{\Gamma \vdash M : \texttt{[}S_1\ S_2\ \cdots S_n\texttt{]} \to T \qquad \Gamma \vdash N_1 : S_1 \qquad \Gamma \vdash S_2 : T_2 \quad \cdots \quad \Gamma \vdash N_n : S_n}{\Gamma \vdash \texttt{(}M\ N_1\ N_2\ \cdots\ N_n\texttt{)} : T}\textsc{T-App}$$

$$\frac{\Gamma \vdash M_1 : T_1 \qquad \Gamma \vdash M_2 : T_2\ \cdots\ \Gamma \vdash M_n : T_n}{\Gamma \vdash \texttt{[}M_1\ M_2\ ...\ M_n\texttt{]} : \texttt{[}T_1\ T_2\ ...\ T_n\texttt{]}}\textsc{T-Tuple}$$

$$\frac{\Gamma \vdash M_1 : T \qquad \Gamma \vdash M_2 : T\ ...\ \Gamma \vdash M_n : T}{\Gamma \vdash \{M_1\ M_2\ ...\ M_n\} : \{T\}}\textsc{T-Collection}$$

$$\frac{\Gamma \vdash C : \texttt{[}S_1\ S_2\ ...S_n\texttt{]} \to T \qquad \Gamma \vdash N_1 : S_1 \qquad \Gamma \vdash S_2 : T_2\ ...\ \Gamma \vdash N_n : S_n}{\Gamma \vdash \texttt{<}C\ N_1\ N_2\ ...\ N_n\texttt{>} : T}\textsc{T-InductiveData}$$

$$\frac{\Gamma \vdash M_1 : T_1 \qquad \Gamma \vdash M_2 : \texttt{(Matcher}\ T_1\texttt{)} \qquad \Gamma \vdash p : \texttt{(Pattern}\ T_1\texttt{)} \qquad \Gamma \mathbin{+\!\!+} V(\Gamma, p) \vdash M_3 : T_3}{\Gamma \vdash \texttt{(match-all}\ M_1\ M_2\ \texttt{[}p\ M_3\texttt{])} : \{T_3\}}\textsc{T-MatchAll}$$

$$\frac{}{\Gamma \vdash \texttt{something} : \texttt{(Matcher}\ T\texttt{)}}\textsc{T-Something}$$

$$\frac{\begin{array}{c}\Gamma \vdash pp_i : \texttt{(PPPattern}\ T\ \texttt{[}S_k\texttt{]}_k\texttt{)} \qquad \Gamma \vdash M_i : \texttt{(Matcher}\ \texttt{[}S_k\texttt{]}_k\texttt{)} \\ \Gamma \vdash dp_{ij} : \texttt{(PDPattern}\ T\texttt{)} \qquad \Gamma \mathbin{+\!\!+} V_{PP}(\Gamma, pp_i) \mathbin{+\!\!+} V_{DP}(\Gamma, dp_i) \vdash N_{ij} : \{\texttt{[}S_k\texttt{]}_k\} \qquad (\forall i, j)\end{array}}{\Gamma \vdash \texttt{(matcher}\ \texttt{[}pp_i\ M_i\ \texttt{[}dp_{ij}\ N_{ij}\texttt{]}_j\texttt{]}_i\texttt{)} : \texttt{(Matcher}\ T\texttt{)}}\textsc{T-Matcher}$$

$$\frac{}{\Gamma \vdash \texttt{\_} : \texttt{(Pattern}\ T\texttt{)}}\textsc{T-Wildcard}$$

$$\frac{}{\Gamma \vdash \texttt{\$}x : \texttt{(Pattern}\ T\texttt{)}}\textsc{T-PatternVariable}$$

$$\frac{\Gamma \vdash M : T}{\Gamma \vdash \texttt{,}M : \texttt{(Pattern}\ T\texttt{)}}\text{T-ValuePattern}$$

$$\frac{\begin{array}{c}\Gamma \vdash C_p : \texttt{[(Pattern}\ S_1\texttt{)}\ \texttt{(Pattern}\ S_2\texttt{)}\ \cdots\ \texttt{(Pattern}\ S_n\texttt{)]} \to \texttt{(Pattern}\ T\texttt{)} \\ \Gamma \vdash M_1 : \texttt{(Pattern}\ S_1\texttt{)} \qquad \Gamma \mathbin{+\!\!+} V(\Gamma, M_1) \vdash M_2 : \texttt{(Pattern}\ S_2\texttt{)} \\ \Gamma \mathbin{+\!\!+} V(\Gamma, M_1) \mathbin{+\!\!+} V((\Gamma \mathbin{+\!\!+} V(\Gamma, M_1)), M_2) \vdash M_3 : \texttt{(Pattern}\ S_3\texttt{)} \\ \cdots \quad \Gamma \mathbin{+\!\!+} V(\Gamma, M_1) \mathbin{+\!\!+} \cdots \mathbin{+\!\!+} V(\Gamma \mathbin{+\!\!+} V(\Gamma, M_1) \mathbin{+\!\!+} \cdots, M_{n-1}) \vdash M_n : \texttt{(Pattern}\ S_n\texttt{)}\end{array}}{\Gamma \vdash \texttt{<}C_p\ M_1\ M_2\ ...\ M_n\texttt{>} : \texttt{(Pattern}\ T\texttt{)}}\textsc{T-InductivePattern}$$

Primitive pattern pattern:

$$\frac{}{\Gamma \vdash \texttt{\$} : \texttt{(PPPattern}\ T\ \texttt{[}T\texttt{])}}\textsc{T-PatternHole}$$

$$\frac{}{\Gamma \vdash \texttt{,\$}x : \texttt{(PPPattern}\ T\ \texttt{[])}}\textsc{T-ValuePatternPattern}$$

$$\frac{\Gamma \vdash C_{pp} : \texttt{[}T_1\ T_2\ ...T_n\texttt{]} \to T \qquad \Gamma \vdash pp_i : \texttt{(PPPattern}\ T_i\ S_i\texttt{)}}{\Gamma \vdash \texttt{<}C_{pp}\ pp_i\texttt{>}_i : \texttt{(PPPattern}\ T\ \Sigma_i S_i\texttt{)}}\textsc{T-InductivePatternPattern}$$

# 項eに型Tがつくとは?

木の根がe:Tである型付け規則を組み合わせた木が存在すること

`(lambda [$x] (b.+ x 10)): Integer -> Integer` の導出木

$$\dfrac{\dfrac{\dfrac{}{\cdots \vdash \texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}}\,\textsc{T-Var} \qquad \dfrac{}{\cdots \vdash x : \texttt{Integer}}\,\textsc{T-Var} \qquad \dfrac{}{\cdots \vdash 10 : \texttt{Integer}}\,\textsc{T-Num}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} +\!\!+ \{x : \texttt{Integer}\} \vdash \texttt{(b.+ x 10)} : \texttt{Integer}}\,\textsc{T-App}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \boxed{\texttt{(lambda [\$x] (b.+ x 10))} : \texttt{Integer} \to \texttt{Integer}}}\,\textsc{T-Abs}$$

# 型付け規則の設計

# Egisonの型システムの概観

- 基本的にはML(TAPL11章)と同じ
- Pattern, Matcherを扱うために
  Pattern型、PPPattern型、PDPattern型、Matcher型を追加
- 非線形パターンとMatcherの中では
  変数のスコープが特殊なため型付け規則が複雑

# Egisonの型一覧

Types

$$
\begin{aligned}
S, T ::= \; & X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool} \\
& \mid T \to T \\
& \mid \texttt{[}T\ T\ \ldots\texttt{]} \\
& \mid \{T\} \\
& \mid \texttt{(Pattern}\ T\texttt{)} \\
& \mid \texttt{(PPPattern}\ T\ T\texttt{)} \\
& \mid \texttt{(PDPattern}\ T\texttt{)} \\
& \mid \texttt{(Matcher}\ T\texttt{)}
\end{aligned}
$$

# Egisonの型一覧

Types

文字列型

$$
\begin{aligned}
S, T ::=\ & X \mid I \mid \boxed{\texttt{String}} \mid \texttt{Integer} \mid \texttt{Bool} \\
& \mid T \to T \\
& \mid \texttt{[}T\ T\ \ldots\texttt{]} \\
& \mid \{T\} \\
& \mid \texttt{(Pattern}\ T\texttt{)} \\
& \mid \texttt{(PPPattern}\ T\ T\texttt{)} \\
& \mid \texttt{(PDPattern}\ T\texttt{)} \\
& \mid \texttt{(Matcher}\ T\texttt{)}
\end{aligned}
$$

例

```
"Hello" :: String
```

# Egisonの型一覧

Types

$$S, T ::= X \mid I \mid \texttt{String} \mid \boxed{\texttt{Integer}} \mid \texttt{Bool}$$

$$\mid T \to T$$

$$\mid [T\ T\ \ldots]$$

$$\mid \{T\}$$

$$\mid (\texttt{Pattern}\ T)$$

$$\mid (\texttt{PPPattern}\ T\ T)$$

$$\mid (\texttt{PDPattern}\ T)$$

$$\mid (\texttt{Matcher}\ T)$$

整数型

例

```
42 :: Integer
```

# Egisonの型一覧

ブール型

Types

$$
\begin{aligned}
S, T ::= &\ X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \boxed{\texttt{Bool}} \\
&\mid T \to T \\
&\mid \texttt{[}T\ T\ \ldots\texttt{]} \\
&\mid \{T\} \\
&\mid \texttt{(Pattern}\ T\texttt{)} \\
&\mid \texttt{(PPPattern}\ T\ T\texttt{)} \\
&\mid \texttt{(PDPattern}\ T\texttt{)} \\
&\mid \texttt{(Matcher}\ T\texttt{)}
\end{aligned}
$$

例

```
#t :: Bool
```

# Egisonの型一覧

Types

関数型

$S, T ::= X \mid T \mid$ `String` $\mid$ `Integer` $\mid$ `Bool`

$\mid \boxed{T \to T}$

$\mid$ [$T$ $T$ ...]

$\mid \{T\}$

$\mid$ (`Pattern` $T$)

$\mid$ (`PPPattern` $T$ $T$)

$\mid$ (`PDPattern` $T$)

$\mid$ (`Matcher` $T$)

例

```
(lambda [$x] (b.+ x 10))
 :: (Integer -> Integer)
```

# Egisonの型一覧

Types

$S, T ::= X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool}$

タプル型

$\mid T \to$

$\mid \boxed{[T\ T\ \ldots]}$

$\mid \{T\}$

$\mid (\texttt{Pattern}\ T)$

$\mid (\texttt{PPPattern}\ T\ T)$

$\mid (\texttt{PDPattern}\ T)$

$\mid (\texttt{Matcher}\ T)$

例

```
[10 2] ::
[Integer Integer]
```

# Egisonの型一覧

Types

$S, T ::= X \mid I \mid$ `String` $\mid$ `Integer` $\mid$ `Bool`

$\mid T \to T$

$\mid [T \ldots]$

コレクション型

$\mid \boxed{\{T\}}$

$\mid$ (`Pattern` $T$)

$\mid$ (`PPPattern` $T$ $T$)

$\mid$ (`PDPattern` $T$)

$\mid$ (`Matcher` $T$)

例

```
{42 1} :: {Integer}
```

# Egisonの型一覧

Types

$$S, T ::= X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool}$$
$$\mid T \to T$$
$$\mid [T\ T\ \ldots]$$
$$\mid \{T\}$$
$$\mid \boxed{(\texttt{Pattern}\ T)}$$
$$\mid (\texttt{PPPattern}\ T\ T)$$
$$\mid (\texttt{PDPattern}\ T)$$
$$\mid (\texttt{Matcher}\ T)$$

Pattern型

例

```
pair ::
([(Pattern Integer) (Pattern Integer)]
  -> (Pattern PairII))
```

# Egisonの型一覧

Types

$$
\begin{aligned}
S, T ::= {} & X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool} \\
& \mid T \to T \\
& \mid [T\ T\ \ldots] \\
& \mid \{T\} \\
& \mid (\texttt{Pattern}\ T) \\
& \mid \boxed{(\texttt{PPPattern}\ T\ T)} \\
& \mid (\texttt{PDPattern}\ T) \\
& \mid (\texttt{Matcher}\ T)
\end{aligned}
$$

PrimitivePatternPattern型

例

# Egisonの型一覧

Types

$$
\begin{aligned}
S, T ::= {} & X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool} \\
& \mid T \to T \\
& \mid \texttt{[}T\ T\ \ldots\texttt{]} \\
& \mid \{T\} \\
& \mid \texttt{(Pattern}\ T\texttt{)} \\
& \mid \texttt{(PPPattern}\ T\texttt{)} \\
& \mid \boxed{\texttt{(PDPattern}\ T\texttt{)}} \\
& \mid \texttt{(Matcher}\ T\texttt{)}
\end{aligned}
$$

PrimitiveDataPattern型

例

# Egisonの型一覧

Types

$$
\begin{aligned}
S, T ::= &\ X \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool} \\
&\mid T \to T \\
&\mid \texttt{[}T\ T\ \ldots\texttt{]} \\
&\mid \{T\} \\
&\mid \texttt{(Pattern}\ T\texttt{)} \\
&\mid \texttt{(PPPattern}\ T\ T\texttt{)} \\
&\mid \texttt{(PDPattern}\ T\texttt{)} \\
&\mid \boxed{\texttt{(Matcher}\ T\texttt{)}}
\end{aligned}
$$

マッチャー型

例

```
something ::
  (Matcher a)
```

# Egisonの型一覧

Types

$$
\begin{aligned}
S, T ::= {} & \boxed{X} \mid I \mid \texttt{String} \mid \texttt{Integer} \mid \texttt{Bool} \\
& \mid T \to T \\
& \mid \texttt{[}T\ T\ \ldots\texttt{]} \\
& \mid \{T\} \\
& \mid \texttt{(Pattern}\ T\texttt{)} \\
& \mid \texttt{(PPPattern}\ T\ T\texttt{)} \\
& \mid \texttt{(PDPattern}\ T\texttt{)} \\
& \mid \texttt{(Matcher}\ T\texttt{)}
\end{aligned}
$$

型変数

例

```
something ::
  (Matcher a)
```

# Egisonの型一覧

Types

$S, T ::= X \mid \boxed{I} \mid$ `String` $\mid$ `Integer` $\mid$ `Bool`

$\mid T \to T$

$\mid$ `[`$T\ T\ \ldots$`]`

$\mid \{T\}$

$\mid$ `(Pattern` $T$

$\mid$ `(PPPattern`

$\mid$ `(PDPattern` $T$`)`

$\mid$ `(Matcher` $T$`)`

ユーザが
定義した
代数的データ型

例

```
Pair ::
([Integer Integer] -> PairII)
```

# 型付け規則の読み方

環境：定義されている変数とその型のリスト

項：だいたいプログラムのこと

$$\boxed{\Gamma} \vdash \boxed{M} : T$$

「環境Γの下で項$M$に型$T$が付く」と読む

結論を導出するための前提条件

規則の名前

$$\frac{\boxed{\Gamma \vdash M_1 : \texttt{Bool} \qquad \Gamma \vdash M_2 : T \qquad \Gamma \vdash M_3 : T}}{\boxed{\Gamma \vdash (\texttt{if}\ M_1\ M_2\ M_3) : T}}\ \boxed{\text{T-IF}}$$

結論

# T-STR (文字列の型付け規則)

$$\frac{}{\Gamma \vdash s : \texttt{String}}\ \textsc{T-Str}$$

前提条件なしで
“Hello” : Stringが導出できたので”Hello”はString型だと言える

使用例

$$\frac{}{\boxed{\epsilon \vdash \texttt{"Hello"} : \texttt{String}}}\ \textsc{T-Str}$$

$$\frac{}{\{x : \texttt{Integer}\} \vdash \texttt{"World!"} : \texttt{String}}\ \textsc{T-Str}$$

# T-If (if式の型付け規則)

$$\frac{\Gamma \vdash M_1 : \texttt{Bool} \qquad \Gamma \vdash M_2 : T \qquad \Gamma \vdash M_3 : T}{\Gamma \vdash (\texttt{if}\ M_1\ M_2\ M_3) : T}\ \textsc{T-If}$$

使用例

$$\dfrac{\dfrac{}{\epsilon \vdash \texttt{true} : \texttt{Bool}}\ \textsc{T-Bool} \qquad \dfrac{}{\epsilon \vdash \texttt{10} : \texttt{Integer}}\ \textsc{T-Num} \qquad \dfrac{}{\epsilon \vdash \texttt{20} : \texttt{Integer}}\ \textsc{T-Num}}{\epsilon \vdash (\texttt{if true 10 20}) : \texttt{Integer}}\ \textsc{T-If}$$

# T-Abs (lambda抽象の型付け規則)

$$\frac{\Gamma \mathbin{+\!\!+} \{x_1 : S_1, x_2 : S_2, ...x_n : S_n\} \vdash M : T}{\Gamma \vdash \texttt{(lambda [\$}x_1\ \texttt{\$}x_2\ ...\ \texttt{\$}x_n\texttt{]}\ M\texttt{)} : \texttt{[}S_1, S_2, ...\texttt{]} \to T}\ \textsc{T-Abs}$$

使用例

$$\dfrac{\dfrac{\vdots}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \mathbin{+\!\!+} \{x : \texttt{Integer}\} \vdash \texttt{(b.+ x 10)} : \texttt{Integer}}\ \textsc{T-App}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : \texttt{[Integer]} \to \texttt{Integer}}\ \textsc{T-Abs}$$

# T-InductivePattern

$$\frac{\begin{array}{c}\Gamma \vdash C_p : [(\texttt{Pattern}\ S_1)\ (\texttt{Pattern}\ S_2)\ \cdots\ (\texttt{Pattern}\ S_n)] \to (\texttt{Pattern}\ T) \\ \Gamma \vdash M_1 : (\texttt{Pattern}\ S_1) \qquad \Gamma +\!\!+\, V(\Gamma, M_1) \vdash M_2 : (\texttt{Pattern}\ S_2) \\ \Gamma +\!\!+\, V(\Gamma, M_1) +\!\!+\, V((\Gamma +\!\!+\, V(\Gamma, M_1)), M_2) \vdash M_3 : (\texttt{Pattern}\ S_3) \\ \cdots \quad \Gamma +\!\!+\, V(\Gamma, M_1) +\!\!+\, \cdots +\!\!+\, V(\Gamma +\!\!+\, V(\Gamma, M_1) +\!\!+\, \cdots, M_{n-1}) \vdash M_n : (\texttt{Pattern}\ S_n)\end{array}}{\Gamma \vdash <\!C_p\ M_1\ M_2\ \ldots\ M_n\!> : (\texttt{Pattern}\ T)}\ \textsc{T-InductivePattern}$$

複雑すぎる…

# 代数的データ型に対するパターンの例

```
> (match-all
      <PairII 10 20>
      (unordered-PairII integer)
      [<pairII $x $y> x])
{10 20}
```

Pattern PairII

# T-InductivePatternは何故複雑になったのか?

非線形パターンが原因でパターン中の変数の有効範囲が複雑

```
pairPP ::
  ([(Pattern PairII) (Pattern PairII)]
    -> (Pattern PairPP))
```

```
<pairPP <pairII $x ,x> <pairII $y ,x>>
```

xの宣言

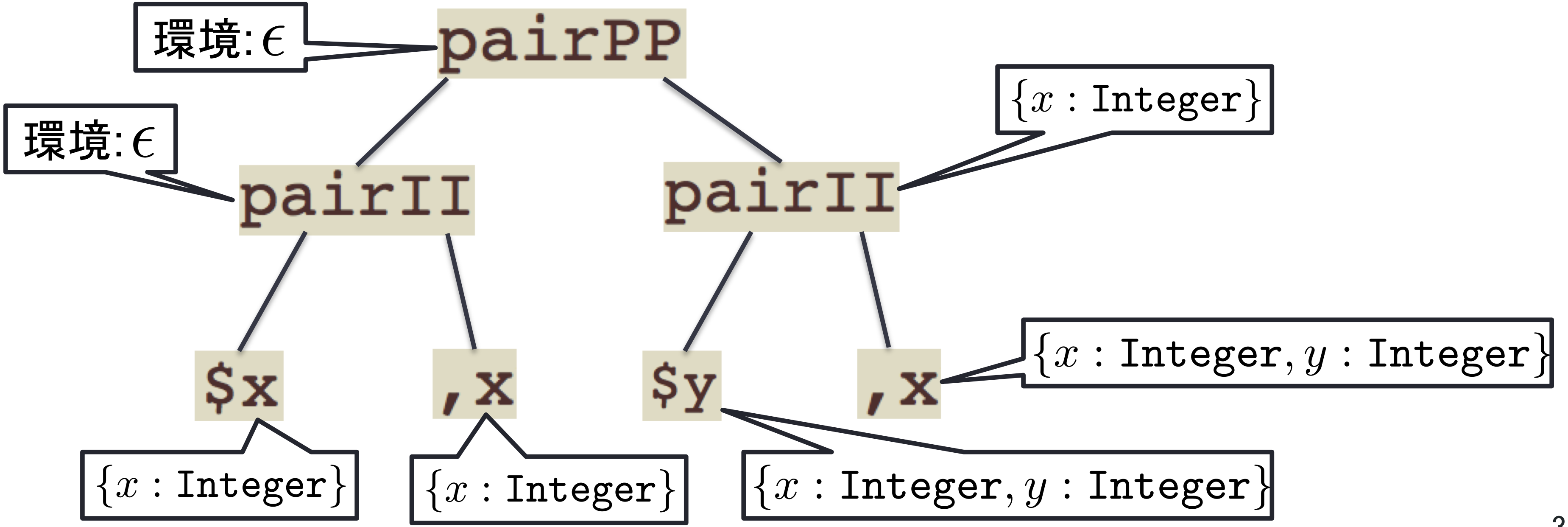
xの使用

xの使用

xの有効範囲

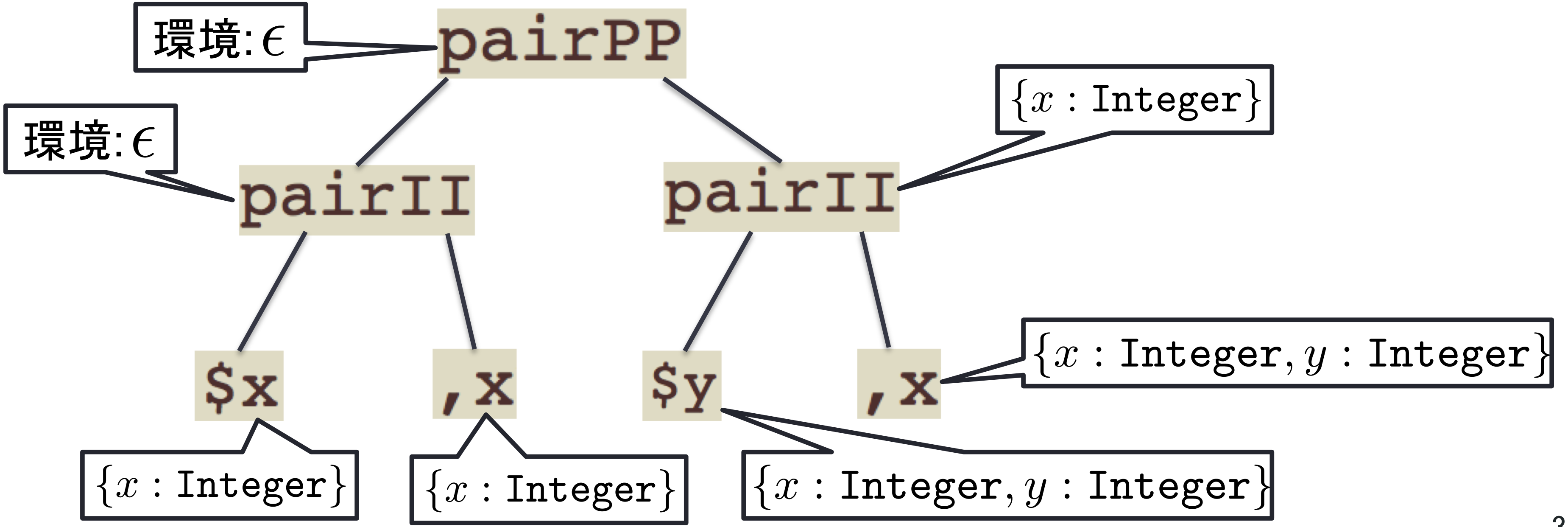
# T-InductivePatternの気持ち

抽象構文木を深さ優先探索しながら環境に変数を追加していく



# T-InductivePatternの気持ち

抽象構文木を深さ優先探索しながら環境に変数を追加していく



# T-Matcher

$$\frac{\Gamma \vdash dp_{ij} : (\texttt{PDPattern}\ \ T) \qquad \begin{array}{c}\Gamma \vdash pp_i : (\texttt{PPPattern}\ \ T\ \ [S_k]_k) \qquad \Gamma \vdash M_i : (\texttt{Matcher}\ \ [S_k]_k) \\ \Gamma \mathbin{+\!\!+} V_{PP}(\Gamma, pp_i) \mathbin{+\!\!+} V_{DP}(\Gamma, dp_i) \vdash N_{ij} : \{[S_k]_k\}\end{array} \qquad (\forall i, j)}{\Gamma \vdash (\texttt{matcher}\ \ [pp_i\ \ M_i\ \ [dp_{ij}\ \ N_{ij}]_j]_i) : (\texttt{Matcher}\ \ T)}\ \textsc{T-Matcher}$$

複雑すぎる…

# T-Matcher は何故複雑になったのか?

各構文要素の対応が複雑

Pattern Integer

Matcher Integer

```
(define $unordered-Pair
 (lambda [$a]
  (matcher {[<pairII $ $> [a a]
   {[<pairII $x $y> {[x y] [y x]}]}]})))
```

Integer

# Egisonに型をつけるインターン

- 型システムの設計

  Egisonの構文の定式化と型付け規則の設計

- 型推論器の実装

  Hindley-Milner型推論アルゴリズムを使った型推論器の実装

# 型推論器とは

与えられた項の型を推論するソフトウェア

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

型推論器

出力：型またはエラー

```
(Integer -> Integer)
```

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮に型変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮に型変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

仮に型変数a,bを置く

$$\frac{}{\{\texttt{b.+} : [\texttt{Integer Integer}] \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : \boxed{a \to b}}\ \textsc{T-Abs}$$

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

$$\dfrac{\dfrac{}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} +\!\!+ \{x : a\} \vdash \texttt{(b.+ x 10)} : b}\ \textsc{T-App}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : a \to b}\ \textsc{T-Abs}$$

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

$$\dfrac{\dfrac{\dfrac{}{\cdots \vdash \texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}}\ \textsc{T-Var} \qquad \dfrac{}{\cdots \vdash x : a}\ \textsc{T-Var} \qquad \dfrac{}{\cdots \vdash 10 : \texttt{Integer}}\ \textsc{T-Num}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} +\!\!+ \{x : a\} \vdash \texttt{(b.+ x 10)} : b}\ \textsc{T-App}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : a \to b}\ \textsc{T-Abs}$$

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

a,bはIntegerと分かる

$$
\dfrac{
  \dfrac{
    \dfrac{}{\cdots \vdash \texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}}\,\textsc{T-Var}
    \quad
    \dfrac{}{\cdots \vdash x : \boxed{a}}\,\textsc{T-Var}
    \quad
    \dfrac{}{\cdots \vdash 10 : \texttt{Integer}}\,\textsc{T-Num}
  }{
    \{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} +\!\!+ \{x : a\} \vdash \texttt{(b.+ x 10)} : \boxed{b}
  }\,\textsc{T-App}
}{
  \{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : a \to b
}\,\textsc{T-Abs}
$$

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

$$
\dfrac{\dfrac{\dfrac{}{\cdots \vdash \texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}}\,\textsc{T-Var} \qquad \dfrac{}{\cdots \vdash x : \texttt{Integer}}\,\textsc{T-Var} \qquad \dfrac{}{\cdots \vdash 10 : \texttt{Integer}}\,\textsc{T-Num}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \mathbin{+\!\!+} \{x : \texttt{Integer}\} \vdash \texttt{(b.+ x 10)} : \texttt{Integer}}\,\textsc{T-App}}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : \texttt{Integer} \to \texttt{Integer}}\,\textsc{T-Abs}
$$

# 型検査のアルゴリズム(Hindley-Milner)

- 型付け規則を下から上に読む
- 分からないところは仮変数を置いて、後で制約を解く

入力：項(プログラム)

```
(lambda [$x] (b.+ x 10))
```

入力された
プログラムの型

$$
\dfrac{
\dfrac{
\dfrac{}{\cdots \vdash \texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}}\,\textsc{T-Var}
\quad
\dfrac{}{\cdots \vdash x : \texttt{Integer}}\,\textsc{T-Var}
\quad
\dfrac{}{\vdash 10 : \texttt{Integer}}\,\textsc{T-Num}
}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} +\!\!+ \{x : \texttt{Integer}\} \vdash \texttt{(b.+ x 10)} : \texttt{Integer}}\,\textsc{T-App}
}{\{\texttt{b.+} : \texttt{[Integer Integer]} \to \texttt{Integer}\} \vdash \texttt{(lambda [\$x] (b.+ x 10))} : \boxed{\texttt{Integer} \to \texttt{Integer}}}\,\textsc{T-Abs}
$$

# Typed Egisonの使い方

- 式の型を調べる

```
> (print-type-of (lambda [$x] (b.+ x 10)))
(lambda [$x] (b.+ x 10)) :: (Integer -> Integer)
```

- 代数的データ構造のコンストラクタと
対応するパターンコンストラクタを定義する

```
> (define-ADT PairII <PairII Integer Integer>)
```

コンストラクタ `PairII` とパターンコンストラクタ `pairII`
が定義される

# 型検査器のデモ

# 今後の課題

- 型健全性の証明
    - Egisonの操作的意味論を小ステップに書き換える
    - 進行性
    - 保存性

# おまけ

- 型なしの言語として設計されたものに型をつけるのは大変
- 実はAny型も入っていてConsistencyぽい感じで型検査をする
    - キャストを挿入しないGradual Typingっぽい感じ?
- [Matcher T Matcher T]がMacther [T T]
  に変換される仕様があり対応が大変

# 参考


- Pierce, Benjamin C., and C. Benjamin. *Types and programming languages*. MIT press, 2002.
- Damas, Luis, and Robin Milner. "Principal type-schemes for functional programs." *Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1982.