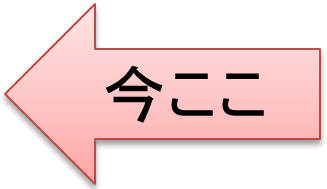
ML 演習 第 6 回

新井淳也、中村宇佑、前田俊行

2011/05/24

言語処理系の作成

- 第 4 回～第 7 回の予定
  - 第 4 回: 基本的なインタプリタの作成
    - 字句解析・構文解析・簡単な言語の処理系の作成
  - 第 5 回: 関数型言語への拡張
    - 関数を作成し呼び出せるように
  - 第 6 回: 言語処理系と型システム
    - ML 風の型推論の実装
  - 第 7 回: インタプリタの様々な拡張
    - 式の評価順序に関する考察



今回の内容

- 型推論 (型システム)
- Unification
- 多相型の扱い

参考文献

- Benjamin C. Pierce,
  "Types and Programming Languages",
  The MIT Press, Cambridge, MA, 2002.

Type System and Type Inference

型システムと型推論

型システムとは?

- プログラム中の式を「型」で分類することで
  プログラムが実行時に不正な動作を
  行わないことを検査・保証する仕組み
  - 「型」とは式の評価結果の値の分類
    - 整数、真偽値、関数など
  - 式が正しい型を持つかどうかをチェックすることを
    「型検査」という

ML 系言語の型システムの特長

- 静的 (static)
  - 型検査はプログラムの実行前に行われる
    - 実行時に型情報を保持する必要が無い
      - つまり実行時のオーバーヘッドがない
- 健全 (sound)
  - 型検査が成功したならば
    そのプログラムは実行時に型エラーを生じない
    - 式を評価した結果は必ず推論で得られた型を持つ
- 型推論 (type inference)
  - ソースコードに型を明示しなくても
    処理系が自動的に式の型を判断して型検査してくれる

型推論規則

- 式がどういう型を持つかを判定する規則
- 例:
  - 整数リテラルは int 型を持つ
  - 真偽値リテラルは bool 型を持つ
  - 式「$E_1 + E_2$」は E_1 と E_2 が共に int 型を持つとき全体として int 型を持つ

簡単な型推論の例

- 式「`0`」の型は?
  - int
- 式「`true`」の型は?
  - bool
- 式「`1 + 2`」の型は?
  - int
- 式「`1 + true`」の型は?
  - 型を持たない (型エラー)

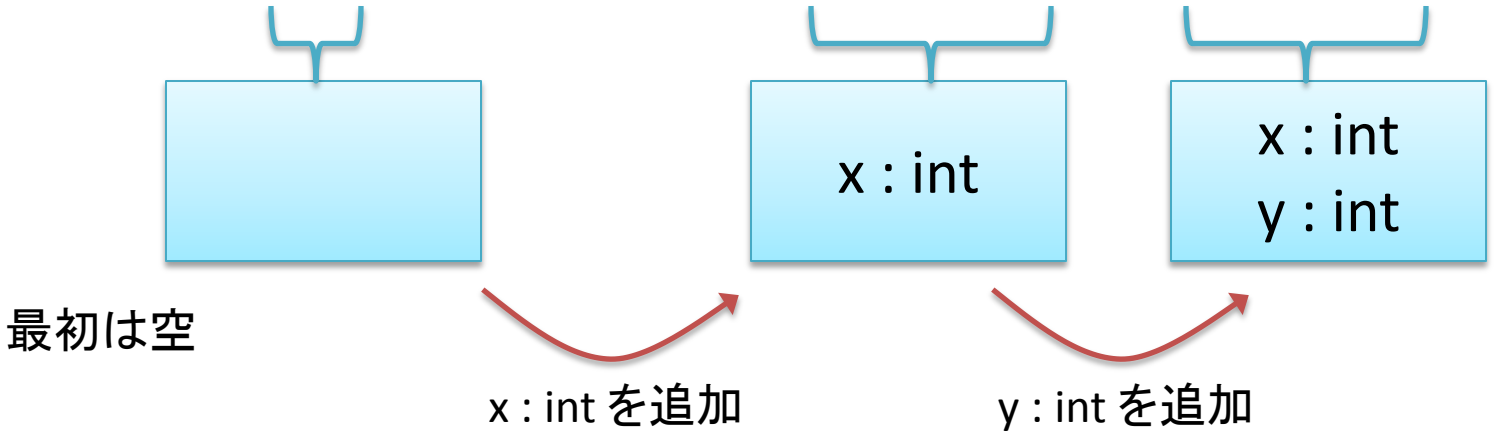
型推論における変数の扱い

- 例えば式「`let x = 1 in x + 2`」を考える
  1. 「1」は int 型を持つので変数 x も int 型を持つ
  2. 「x」が int 型を持ち「2」も int 型を持つので
     「x + 2」も int 型を持つ
  3. よって「let …」全体も int 型を持つ
- つまり「x + 2」を型検査するとき
  変数 x が int 型を持つこと
  を覚えていないといけない

型環境

- 変数の型を「型環境」として覚えておく
  - 型環境 = 変数から型への写像

```
let x = 1 in let y = x + 2 in x + y
```



let 式の型推論規則

- 式「let $I = E_1$ in E_2」の型推論は以下の通り
  - 現在の型環境で式 E_1 を型推論する
    - 得られた式 E_1 の型を T とする
  - 型環境に「$I : T$」を加える
  - 拡張された型環境で式 E_2 を型推論する
    - 式 E_2 の型が let 式全体の型になる

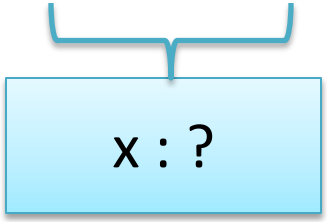
変数の型推論規則

- 式「x」の型推論は以下の通り
  - 型環境に「x : *T*」 が含まれていたら「x」の型は *T*
    - そうでなければ「x」は型を持たない (型エラー)

λ抽象の型推論について

- 式「fun x -> x + 1」の型は?
  - 「x + 1」の型が関数の戻り値の型となるが……
  - 「x + 1」の型検査を始める段階では
    まだ束縛変数 x の型が分からない!

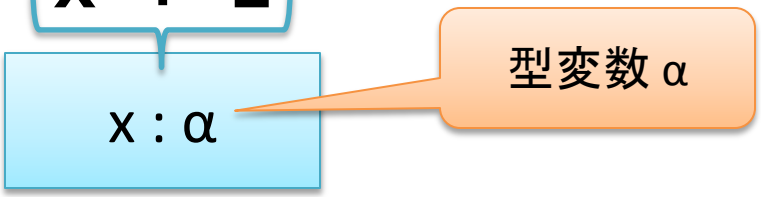
```
fun x -> x + 1
```



型変数と制約

- とりあえず適当な型で
  型環境を拡張して型検査する
  - 適当な型を「型変数」とする

```
fun x -> x + 1
```



- 「x + 1」の型検査では
  α が int でなければならないことがわかる
  - そこで型推論では推論結果の型 (int) だけでなく
    「α = int」という制約も生成することにする
    - 生成された制約は後で解決する

関数適用の型推論

- 式「$E_1\ E_2$」の型は?
  - まず式 E_1, E_2 の型推論をする
    - 式 E_1, E_2 の型をそれぞれ T_1, T_2 とする
  - 関数適用のためには
    T_1 は T_2 を引数とする関数型でないといけない
  - 関数の返り値の型はまだ分からない
    - そこで、戻り値の型として新たな型変数 α を導入し
      α を式「$E_1\ E_2$」の型とするとともに
      「$T_1 = T_2 \rightarrow \alpha$」という制約も生成する

let rec 式の型推論規則

- 式「let rec $I = E_1$ in E_2」の型推論は以下の通り
  - 新たな型変数αを導入して型環境を拡張する
    - 型環境に「$I : \alpha$」を加える
  - 拡張した型環境で式 E_1 を型推論する
    - 得られた式 E_1 の型を T とする
  - 新たに制約「$T = \alpha$」を生成する
  - 拡張された型環境で式 E_2 を型推論する
    - 式 E_2 の型が let 式全体の型になる

型推論の例: 制約の生成まで

- 以下の式の型推論を考える

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n – 1) in fact 10
```

型推論の例: 制約の生成まで

$\Gamma \vdash e : t, C$

読み方:
型環境 Γ の下で式 e を型推論すると
その型は t で制約 C を満たす必要がある

$$\frac{fact:\alpha \vdash \texttt{fun n -> } \ldots : t_1, C_1 \qquad fact:\alpha \vdash \ldots : t_2, C_2}{\vdash \texttt{let rec fact n = } \ldots \texttt{ in } \ldots : t_2, C_1 \cup C_2 \cup \{t_1 = \alpha\}}$$

推論規則:
これらが
成り立つ時...

これが
成り立つ
と読む

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{\texttt{n:}\beta,\ \texttt{fact:}\alpha \vdash \texttt{if … then … else … : t, C}}{\texttt{fact:}\alpha \vdash \texttt{fun n -> … : }\beta\texttt{→t, C}}}{\cdots}$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\mathtt{n}:\beta,\ \mathtt{fact}:\alpha \vdash \mathtt{n} = 0 : \mathtt{bool},\ C_b \qquad \ldots \qquad \ldots}{\mathtt{n}:\beta,\ \mathtt{fact}:\alpha \vdash \mathtt{if}\ldots\ \mathtt{then}\ldots\ \mathtt{else}\ldots : t_t,\ C_b \cup C_t \cup C_e \cup \{t_t = t_e\}}$$

$$\ldots$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{\dfrac{}{n : \beta \in n{:}\beta,\ fact{:}\alpha}}{n{:}\beta,\ fact{:}\alpha \vdash n : \beta,\ \{\}} \qquad n{:}\beta,\ fact{:}\alpha \vdash 0 : int,\ \{\}}{n{:}\beta,\ fact{:}\alpha \vdash n = 0 : bool,\ \{\beta{=}int\}}$$

…

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\cdots \qquad n:\beta,\ fact:\alpha \vdash 1 : int,\ \{\} \qquad \cdots}{n:\beta,\ fact:\alpha \vdash \texttt{if}\ldots\ \texttt{then}\ldots\ \texttt{else}\ldots : int,\ \{\beta=int\} \cup C_e \cup \{int=t_e\}}$$

$$\cdots$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\cdots \quad \cdots \quad n:\beta,\ fact:\alpha \vdash n * fact\ (n - 1):\ t_e,\ C_e}{n:\beta,\ fact:\alpha \vdash if\ldots\ then\ldots\ else\ldots\ :\ int,\ \{\beta=int\} \cup C_e \cup \{int=t_e\}}$$

$$\cdots$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{n : \beta \in n{:}\beta,\ fact{:}\alpha}{n{:}\beta,\ fact{:}\alpha \vdash n : \beta,\ \{\}} \qquad n{:}\beta, fact{:}\alpha \vdash fact\ (n - 1) : t_r,\ C_r}{n{:}\beta,\ fact{:}\alpha \vdash n * fact\ (n - 1) : int,\ \{\beta = int\} \cup C_r \cup \{t_r = int\}}$$

…

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{n:\beta,\ fact:\alpha \vdash fact : t_c, C_c \qquad n:\beta, fact:\alpha \vdash n-1 : t_a, C_a}{\dfrac{n:\beta,\ fact:\alpha \vdash fact\ (n-1) : \gamma,\ C_c \cup C_a \cup \{t_c = t_a \rightarrow \gamma\}}{\ldots}}$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{\mathtt{fact} : \alpha \in \mathtt{n}{:}\beta, \mathtt{fact}{:}\alpha}{\mathtt{n}{:}\beta, \mathtt{fact}{:}\alpha \vdash \mathtt{fact} : \alpha, \{\}} \quad \mathtt{n}{:}\beta, \mathtt{fact}{:}\alpha \vdash \mathtt{n-1} : t_a, C_a}{\mathtt{n}{:}\beta, \mathtt{fact}{:}\alpha \vdash \mathtt{fact\ (n-1)} : \gamma, C_a \cup \{\alpha = t_a \rightarrow \gamma\}}$$

…

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n – 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{n : \beta \in n{:}\beta,\ fact{:}\alpha}{n{:}\beta,\ fact{:}\alpha \vdash n : \beta,\{\}} \qquad n{:}\beta,\ fact{:}\alpha \vdash 1 : int,\{\}}{\dfrac{n{:}\beta,\ fact{:}\alpha \vdash n - 1 : int,\ \{\beta{=}int\}}{\cdots} \cup \{int{=}int\}}$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{n:\beta, fact:\alpha \vdash fact:\alpha, \{\} \quad n:\beta, fact:\alpha \vdash n-1:int, \{\beta=int\}}{n:\beta,\ fact:\alpha \vdash fact\ (n\ -\ 1) : \gamma, \{\beta=int\} \cup \{\alpha=int \rightarrow \gamma\}}}{\ldots}$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{}{n{:}\beta,\ fact{:}\alpha \vdash n : \beta,\ \{\}} \qquad \dfrac{}{n{:}\beta, fact{:}\alpha \vdash fact\ (n - 1) : \gamma, \{\beta = int\} \cup \{\alpha = int \to \gamma\}}}{n{:}\beta,\ fact{:}\alpha \vdash n * fact\ (n - 1) : int,\ \{\beta = int\} \cup \{\alpha = int \to \gamma\} \cup \{\gamma = int\}}$$

…

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\dots \quad \dots \qquad n:\beta,\ fact:\alpha \vdash n * fact\ (n - 1) : int,\ \{\beta=int\}\cup\{\alpha=int\to\gamma\}\cup\{\gamma=int\}}{n:\beta,\ fact:\alpha \vdash if\dots\ then\dots\ else\dots : int,\ \{\beta=int\}\cup\{\alpha=int\to\gamma\}\cup\{\gamma=int\}\cup\{\cancel{int=int}\}}$$

$$\dots$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\dfrac{n:\beta,\ fact:\alpha \vdash \texttt{if…\ then…\ else…} : int,\ \{\beta=int\}\cup\{\alpha=int\to\gamma\}\cup\{\gamma=int\}}{fact:\alpha \vdash \texttt{fun\ n\ ->\ …} : \beta\to int,\ \{\beta=int\}\cup\{\alpha=int\to\gamma\}\cup\{\gamma=int\}}}{\ldots}$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\mathtt{fact}:\alpha \vdash \mathtt{fun\ n\ ->\ \ldots\ :\ }\beta\to\mathtt{int},\ \{\beta=\mathtt{int}\}\cup\{\alpha=\mathtt{int}\to\gamma\}\cup\{\gamma=\mathtt{int}\} \quad \mathtt{fact}:\alpha \vdash \ldots : t_2,\ C_2}{\vdash \mathtt{let\ rec\ fact\ n\ =\ \ldots\ in\ \ldots\ :}\ t_2,\ \{\beta=\mathtt{int}\}\cup\{\alpha=\mathtt{int}\to\gamma\}\cup\{\gamma=\mathtt{int}\}\cup C_2\cup\{\beta\to\mathtt{int}=\alpha\}}$$

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

型推論の例: 制約の生成まで

$$\dfrac{\dfrac{\dfrac{}{\mathtt{fact} : \alpha \in \mathtt{fact}:\alpha}}{\mathtt{fact}:\alpha \vdash \mathtt{fact} : \alpha, \{\}} \qquad \mathtt{fact}:\alpha \vdash 10 : \mathtt{int}, \{\}}{\mathtt{fact}:\alpha \vdash \mathtt{fact}\ 10 : \delta, \{\alpha = \mathtt{int} \to \delta\}}$$

…

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n – 1) in fact 10
```

型推論の例: 制約の生成まで

$$\frac{\mathtt{fact}:\alpha \vdash \mathtt{fun\ n\ ->\ \ldots\ :\ }\beta\to\mathtt{int},\ \{\beta=\mathtt{int}\}\cup\{\alpha=\mathtt{int}\to\gamma\}\cup\{\gamma=\mathtt{int}\} \qquad \mathtt{fact}:\alpha \vdash \mathtt{fact\ 10\ :\ }\delta,\ \{\alpha=\mathtt{int}\to\delta\}}{\vdash \mathtt{let\ rec\ fact\ n\ =\ \ldots\ in\ \ldots\ :\ }\delta,\ \{\beta=\mathtt{int}\}\cup\{\alpha=\mathtt{int}\to\gamma\}\cup\{\gamma=\mathtt{int}\}\cup\{\alpha=\mathtt{int}\to\delta\}\cup\{\beta\to\mathtt{int}=\alpha\}$$

あとはこの制約を解くだけ

- 型推論対象の式:

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n - 1) in fact 10
```

Constraint Solving and Unification

制約の解決と UNIFICATION

型の制約

- 型推論の結果として式の型だけでなく型に関する制約も返される
  - 例えば式「fun x -> x + 1」の型検査の結果として
    - 型「α→int」と
    - 制約「α = int」が得られる
      - この制約を解決すると最終的な型として「int → int」が得られる

Unification

- 与えられた制約を満たすような
  型変数の置き換え方を求めること
  - 入力: 制約 (「$T_1 = T_2$」の形の等式の集合)
  - 出力: 型変数から型への写像
    - 型推論で得られた型をこの出力で substitute することで最終的な型が得られる

Unification の例

- Unify({ α = int })
  = { α ⇒ int }

- Unify({ α → β = bool → γ })
  = { α ⇒ bool, β ⇒ γ }

- Unify({ α → β = int })
  → unification 不可

Unification アルゴリズム

- Unify({ }) = { }
- Unify({ $T = T$ } ∪ C) = Unify(C)
- Unify({ $T_{11} \rightarrow T_{12} = T_{21} \rightarrow T_{22}$ } ∪ C) =
  Unify({ $T_{11} = T_{21}$ } ∪ { $T_{12} = T_{22}$ } ∪ C)
- Unify({ $\alpha = T$ } ∪ C) = Unify({ $T = \alpha$ } ∪ C) =
  Unify($[\alpha \Rightarrow T]C$)∘ { $\alpha \Rightarrow T$ }
  - » ただし T の自由変数に α が含まれない場合
- Unify({ その他 }) ⇒ Unification 不可
  - ただし
    - $[\alpha \Rightarrow T]C$ は C 中の自由な α を T に置き換えたものを表す
    - $(S_1 \circ S_2)\ T$ は $S_1\ (S_2\ T)$ を表す

型推論の例の続き: unification

- 以下の式の型推論の続き

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n – 1) in fact 10
```

- 型推論規則の結果:
  - 型 : **δ**
  - 制約:**{β=int}**∪**{α=int→γ}**∪**{γ=int}**
    ∪**{α=int→δ}**∪**{β→int=α}**

型推論の例の続き: unification

- 制約の unification
  - Unify ($\{\beta=int\} \cup \{\alpha=int\to\gamma\} \cup \{\gamma=int\} \cup \{\alpha=int\to\delta\} \cup \{\beta\to int=\alpha\}$)

➡ Unify ($\{\alpha=int\to\gamma\} \cup \{\gamma=int\} \cup \{\alpha=int\to\delta\} \cup \{int\to int=\alpha\}) \circ \{\beta \Rightarrow int\}$

➡ Unify ($\{\gamma=int\} \cup \{int\to\gamma=int\to\delta\} \cup \{int\to int=int\to\gamma\}) \circ \{\alpha \Rightarrow int\to\gamma\} \circ \{\beta \Rightarrow int\}$

➡ Unify ($\{int\to int=int\to\delta\} \cup \{int\to int=int\to int\}$) $\circ \{\gamma \Rightarrow int\} \circ \{\alpha \Rightarrow int\to\gamma\} \circ \{\beta \Rightarrow int\}$

➡ $\{\delta \Rightarrow int\} \circ \{\gamma \Rightarrow int\} \circ \{\alpha \Rightarrow int\to\gamma\} \circ \{\beta \Rightarrow int\}$

型推論の例の続き: unification

- 以下の式の型推論の続き

```
let rec fact n =
  if n = 0 then 1
  else n * fact (n – 1) in fact 10
```

- 型推論規則と unification の結果:
  - 制約の解:
    S = {δ ⇒ int}∘{γ ⇒ int}∘{α ⇒ int→γ}∘{β ⇒ int}
  - 型 : S(δ) = int

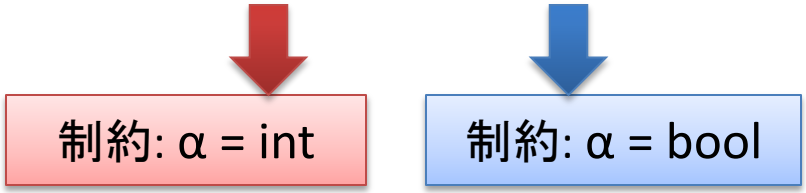
Inference of Polymorphic Types

多相型の型推論

ここまでの型推論方法の問題点

- OCaml 風の多相型を実現することができない
  - 式「fun x -> x」の推論結果は型「α → α」だが このαは「未決定な単相型」(第2回資料参照)
- 上手くいかない例:

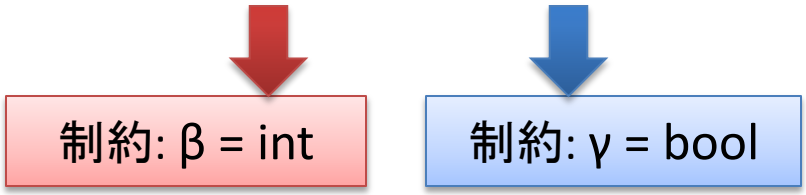
```
let id = fun x -> x in (id 0, id true)
```



  - 「α = int」と「α = bool」の二つの制約を 同時に満たす解はない

OCaml 風の多相型を実現する手法

- 「型スキーマ」の概念を導入する
  - 「型スキーマ」=「型 t」と
    「その型中の未決定な型変数のうち
    多相的に扱うものの集合 Δ」の組
    - ∀Δ . t のように書く
- 例: `let id = fun x -> x in (id 0, id true)`



  - 変数 id の型スキーマを∀α.α → α とおく
    - 一つ目の変数 id の出現では新たな型変数 β で
      二つ目の変数 id の出現では新たな型変数 γ で
      型スキーマを instantiate すればよい

型スキーマを導入する方法

- OCaml 風の型推論では以下の通り
  - 前準備
    - 型環境を変数から型スキーマへの写像とする
  - 型スキーマの作成のタイミング
    - let 式で変数を束縛するときに
      型スキーマを作成して型環境に加える
  - 型スキーマの instantiate のタイミング
    - 変数が式に現れた時には型スキーマを
      instantiate して実際の型に置き換える

let 式の型推論規則 (改)

- 式「let $I = E_1$ in E_2」の型推論規則
  - 現在の型環境 Γ で式 E_1 を型推論する
    - 式 E_1 の型を T、また生成された制約を C とする
    - 制約 C の解を S (= Unify(C))、また $T' = S(T)$ とする
    - Γ' を Γ に現れる全ての型を S で置き換えたものとする
      - ただし、既に多相的に扱っている型変数は置き換えない
  - 型環境 Γ' に「$I : \forall P . T'$」を加える
    - P は T' の中で多相的に扱える型変数の集合
      - P = (T' に含まれる型変数の集合) – (型環境に含まれている型変数の集合)
  - 拡張された型環境で式 E_2 を型推論する
    - 式 E_2 の型が let 式全体の型になる

変数の型推論規則 (改)

- 式「x」の型推論規則
  - 型環境に「$x : \forall P\,.\,T$」が含まれていたら
    「x」の型は T' とする
    - ただし T' は T 中に現れる型変数のうち
      P に含まれるものを新しい型変数で置き換えたもの

多相型の型推論の例

- 以下の式の型推論を考える

```
let f x = x in f f
```

多相型の型推論の例

$$\frac{\vdash \texttt{fun x -> x} : t_1, C_1 \qquad f : \forall P.t_1' \vdash \texttt{f f} : t_2, C_2}{\vdash \texttt{let f x = x in f f} : t_2, C_1 \cup C_2}$$

ただし
t_1'は $\mathrm{Unify}(C_1)(t_1)$、
またPはt_1'に現れる
自由な型変数のうち
型環境に自由に現れるものを
除いたものの集合とする

- 型推論対象の式:

```
let f x = x in f f
```

多相型の型推論の例

$$\dfrac{\dfrac{x : \forall\{\}.\alpha \in x : \forall\{\}.\alpha}{x : \forall\{\}.\alpha \vdash x : \alpha, \{\}} \quad \vdash \texttt{fun x -> x} : \alpha\to\alpha, \{\} \quad f : \forall P.t_1' \vdash f\ f : t_2, C_2}{\vdash \texttt{let f x = x in f f} : t_2, \{\} \cup C_2}$$

ただし
t_1'は $\mathrm{Unify}(C_1)(t_1)$、
またPはt_1'に現れる
自由な型変数のうち
型環境に自由に現れるものを
除いたものの集合とする

- 型推論対象の式:

```
let f x = x in f f
```

多相型の型推論の例

$$\dfrac{\dfrac{x : \forall\{\}.\alpha \in x : \forall\{\}.\alpha}{x : \forall\{\}.\alpha \vdash x : \alpha, \{\}} \quad \vdash \texttt{fun x -> x} : \alpha\to\alpha, \{\} \quad \texttt{f} : \forall\alpha.\alpha\to\alpha \vdash \texttt{f f} : t_2, C_2}{\vdash \texttt{let f x = x in f f} : t_2, C_2}$$

- 型推論対象の式:

```
let f x = x in f f
```

多相型の型推論の例

$$\dfrac{\dfrac{f:\forall\alpha.\alpha\to\alpha \in f:\forall\alpha.\alpha\to\alpha}{f:\forall\alpha.\alpha\to\alpha \vdash f:\beta\to\beta,\{\}} \quad \dfrac{f:\forall\alpha.\alpha\to\alpha \in f:\forall\alpha.\alpha\to\alpha}{f:\forall\alpha.\alpha\to\alpha \vdash f:\gamma\to\gamma,\{\}}}{\dfrac{\ldots \qquad f : \forall\alpha.\alpha\to\alpha \vdash f\ f : t_2, C_2}{\vdash \texttt{let f x = x in f f} : t_2, C_2}}$$

- 型推論対象の式:

```
let f x = x in f f
```

多相型の型推論の例

$$\dfrac{\dfrac{f:\forall\alpha.\alpha\to\alpha \in f:\forall\alpha.\alpha\to\alpha}{f:\forall\alpha.\alpha\to\alpha \vdash f:\beta\to\beta,\{\}} \quad \dfrac{f:\forall\alpha.\alpha\to\alpha \in f:\forall\alpha.\alpha\to\alpha}{f:\forall\alpha.\alpha\to\alpha \vdash f:\gamma\to\gamma,\{\}}}{\dfrac{\ldots \quad f : \forall\alpha.\alpha\to\alpha \vdash f\ f : \delta,\{\beta\to\beta=(\gamma\to\gamma)\to\delta\}}{\vdash \texttt{let f x = x in f f} : \delta,\{\beta\to\beta=(\gamma\to\gamma)\to\delta\}}}$$

- 型推論対象の式:

```
let f x = x in f f
```

多相型の型推論の例

$$\frac{\dots}{\vdash \texttt{let f x = x in f f} : \delta, \{\beta \to \beta = (\gamma \to \gamma) \to \delta\}}$$

$$\mathrm{Unify}(\{\beta \to \beta = (\gamma \to \gamma) \to \delta\}) = \{\delta \Rightarrow \gamma \to \gamma\} \circ \{\beta \Rightarrow \gamma \to \gamma\}$$

- 型推論対象の式:

  let f x = x in f f

このやり方ではうまくいかない例

- 下の式は型推論できない

```
(fun f -> (f 0, f true)) (fun x -> x)
```

  - 「`f 0`」は「(f の型) = int → α」という制約を返すが
    「`f true`」は「(f の型) = bool → β」という制約を返す

第 6 回 課題

締切: 6/7 13:00 (日本標準時)

課題 1 (5点)

- 第5回の課題のインタプリタが扱う値に応じて型 **typ** を定義せよ
  - 整数型、真偽値型、関数型、型変数を含めること

$$T \;=\; Int \mid Bool \mid T \rightarrow T \mid \alpha$$

    - 型変数を表す型 **type_var** も別途定義するとよい
  - 必要に応じてリストやタプルの型も含めること

$$T \;=\; \ldots \mid (T, T) \mid T\ List$$

課題 2 (5点)

- 型変数 α と二つの型 T, T' を受け取り、T の中に現れる α を全て T' に置き換える関数 **subst_typ** を定義せよ

```
subst_typ : typ -> type_var -> typ -> typ
```

  - 引数の順番は自由

課題 3 (10点)

- Unification アルゴリズムを実装する関数 **unify** を定義せよ

```
unify : cstrts -> substitution
```

  - 例えば

```
type cstrts = (typ * typ) list
type substituion = (type_var * typ) list
```

    と表すと **unify** は
    制約 (型のペアのリスト) を受け取り
    型変数と型のペアのリストを返す関数となる
    - もちろん他の表現方法でもよい

課題 4 (15点)

- 第5回の課題のインタプリタを拡張して
  多相型なし・パターンマッチングなしの
  型推論を行うようにせよ
  - 例えば以下のような関数を定義することになるはず
    - `type_check_expr :`
      `type_env -> expr -> typ * cstrts`
    - `type_check_cmd :`
      `type_env -> cmd -> type_env * typ`

      (* 制約を解いて推論結果の型に反映させてから
      次のコマンドを処理する *)

課題４の注意

- 新しい型変数を導入する際
  他の型変数とかぶらないようにすること
  - 呼び出すたびに毎回異なる型変数を返す関数を用意しておくとよい

    `new_type_var : unit -> type_var`
    - 参照を用いると容易に実装できる

課題 5 (15点)

- インタプリタを拡張して
  型推論を多相型に対応させよ
  - 例えば以下のような関数を定義するとよいはず
    - `generalize : type_env -> typ -> type_schema`
    - `instantiate : type_schema -> typ`
    - `get_type_vars : typ -> type_vars`

課題 6 (15点)

- インタプリタを拡張して
  パターンマッチングの型推論に対応させよ
  - 例えば以下のような関数を定義することになるはず
    - `type_check_pattern :`
      `type_env -> pattern`
      `-> type_env * typ * cstrts`

課題 7 (25点)

- 今回実装した型推論が
  健全であることを証明せよ
  - 即ち、
    型推論でエラーにならなかった式は
    評価 (実行) してもインタプリタ上で
    評価時エラーを生じないことを証明せよ
    - 証明できない場合は
      型推論規則を変更するなどした上で証明せよ

課題 8 (25点)

- インタプリタを拡張して
  以下の式を型推論できるようにせよ
  - 型推論の健全性や停止性に留意すること

  - `(fun f -> f true && f 1 = 0) (fun x -> x)`

課題 9 (25点)

- インタプリタを拡張して
  以下の再帰関数を型推論できるようにせよ
  - 型推論の健全性や停止性に留意すること

```
– let rec tet = fun n f z ->
      if n <= 0 then z
      else tet (n - 1) (fun v w -> v (v w)) f z ;;
```

課題 10 (25点)

- インタプリタを拡張して
  関数 f を受け取ってf を再帰的に無限回合成する関数を返す関数 fix を定義できるようにせよ
  - f は「関数を受け取って関数を返す関数」と仮定してよい
  - 以下のようなイメージ

```
fix f ≡ f (f (f (f (f (f (f ...))))))
```

  - 使用例

```
# let fib = fix (fun g n ->
                 if n <= 2 then 1
                 else g (n - 1) + g (n - 2));;
val fib : int -> int = <fun>
# fib 10;;
- : int = 55
```

  - ただし、fix は let rec を用いずに定義し
    型検査にパスするようにすること
    - 型推論の健全性や停止性に留意すること

課題 11 (25点)

- インタプリタを拡張して
  以下の式を型推論できるようにせよ
  - 型推論の健全性や停止性に留意すること

```
let x = fun x -> (x, x) in
let x = fun y -> x (x y) in
let x = fun y -> x (x y) in
let x = fun y -> x (x y) in
let x = fun y -> x (x y) in
let x = fun y -> x (x y) in
x (fun x -> x)
```