ML 演習 第 7 回

2007/07/17

飯塚 大輔, 後藤 哲志, 前田 俊行

http://www.yl.is.s.u-tokyo.ac.jp/~sgotou/lecture/caml-enshu

今回の内容

- 言語処理系の実装 (3)
  - 型推論
    - 型推論の例
    - Unification
    - 多相型の扱い

型推論とは?

- 型を推論すること
  - 「型」
    式を評価した結果どんな値になるかを表すもの
    - 整数、論理値、関数、etc.
  - 「推論」
    与えられた式から型を推定すること

ML系言語の型推論の特長

- 静的 (static)
  - 式を見るだけで型が決定する
    - 実行時に型情報を保持する必要が無い
- 健全 (sound)
  - 式を推論して型が得られたならば
    その式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
- つまり型推論に成功した ML プログラムは
  実行時に型エラーを生じない上
  実行時の型検査も不要
    - 今回は MiniML に ML 風型推論を実装してみる

MiniML の型

- 今回扱う型

τ ::=	int	整数
|	bool	論理値
|	τ_1 * τ_2	ペア
|	τ list	リスト
|	$\tau_1 \rightarrow \tau_2$	関数 (定義域 → 値域)

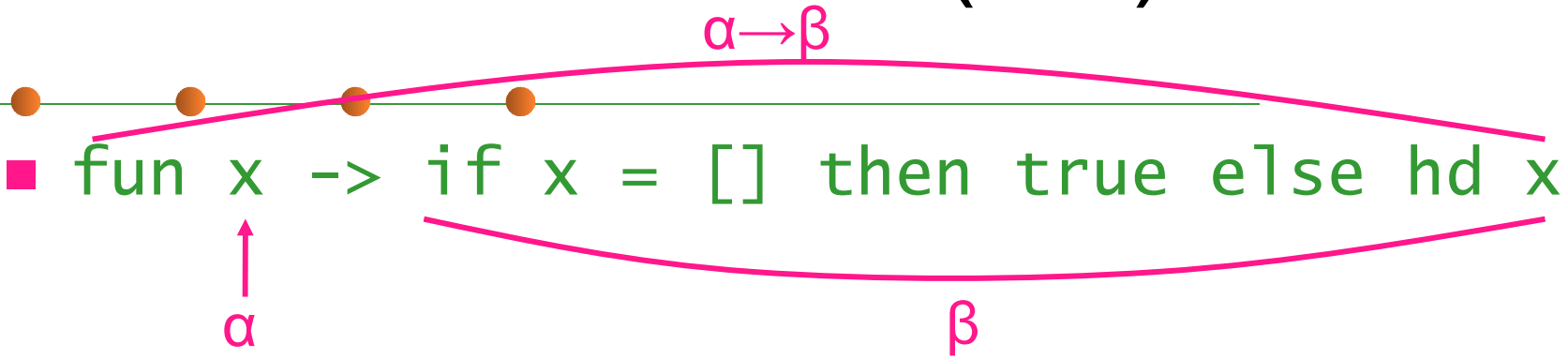
型付けの例 (1/7)

- `fun x -> if x = [] then true else hd x`

  (hd : τ_{hd} := τ list → τ)

  - { hd: τ_{hd} } の下で fun ... の型を考える

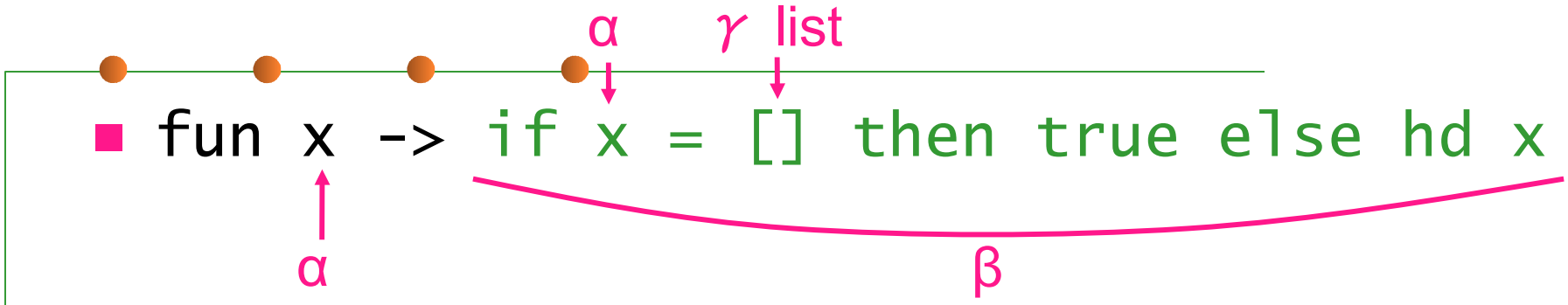
型付けの例 (2/7)



(hd : τ_{hd} := τ list → τ)

- { hd: τ_{hd} } の下で fun ... の型を考える
- 関数なので (fun x -> ...): α → β と置く
- 引数の型と見比べると x : α
- { hd: τ_{hd}, x: α } の下で (if ...) : β
- { hd: τ_{hd}, x: α } の下で if ... の型を調べる

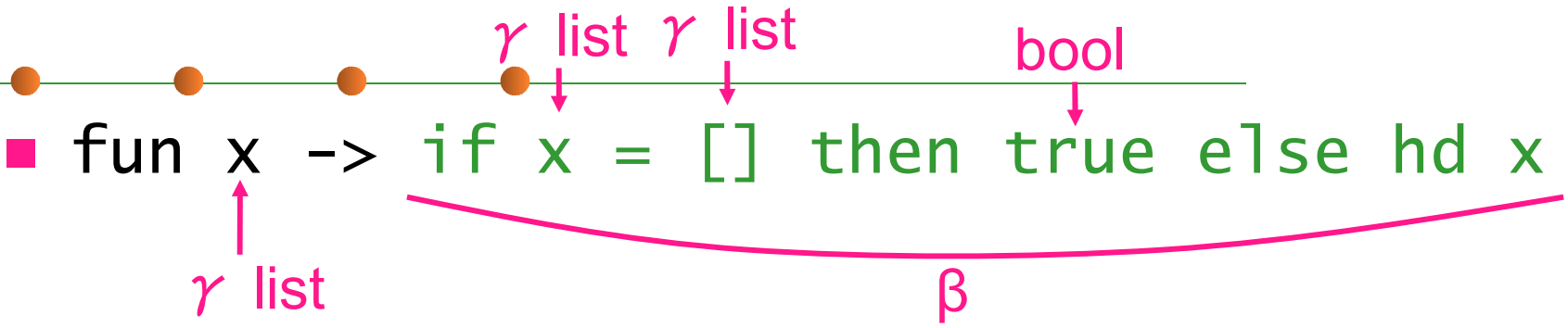
型付けの例 (3/7)



(hd : τ_{hd} := τ list → τ)

- { hd: τ_{hd}, x: α } の下で if ... の型を調べる
- if の条件節 x = [] より α = γ list

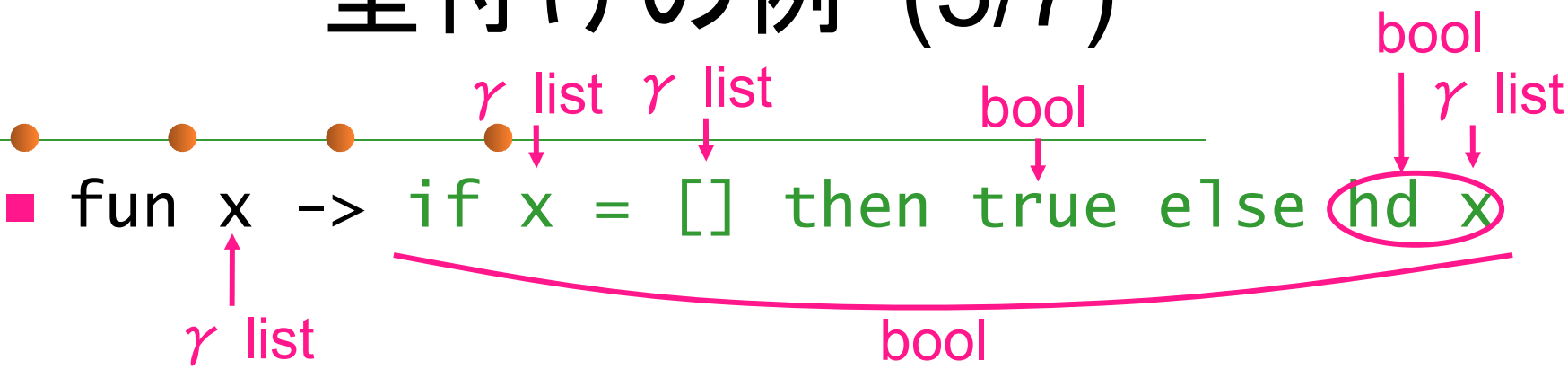
型付けの例 (4/7)



(hd : τ_{hd} := τ list → τ)

- { hd: τ_{hd}, x: α } の下で if ... の型を調べる
- if の条件節 x = [] より α = γ list
- then 節 true より (if ...) : bool,
  hd x : bool

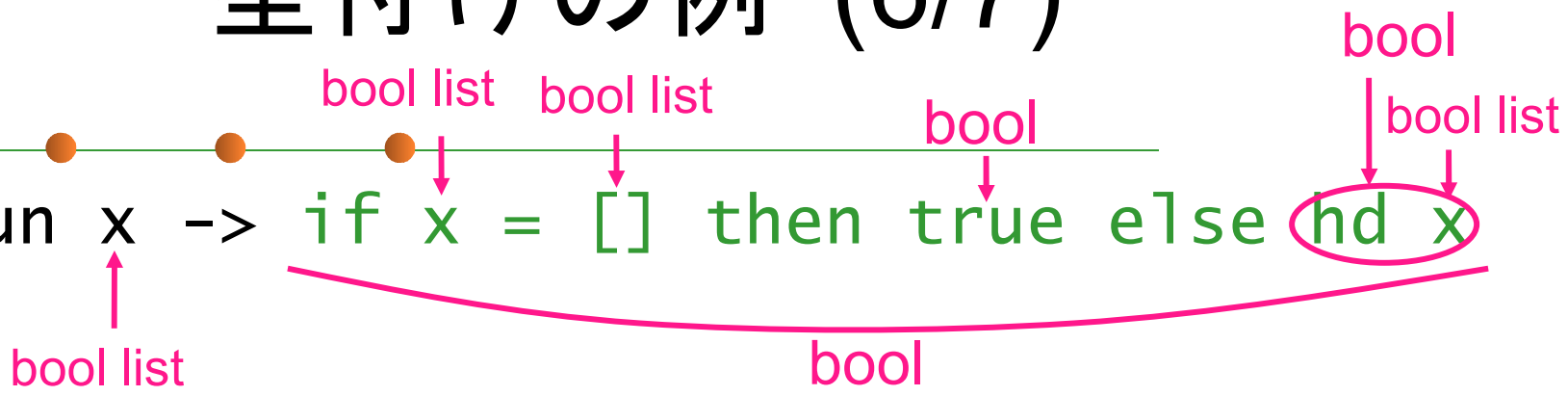
型付けの例 (5/7)



(hd : τ_{hd} := τ list → τ)

- { hd: τ_{hd}, x: α } の下で if ... の型を調べる
- if の条件節 x = [] より α = γ list
- then 節 true より (if ...) : bool
- hd: τ list → τ と
  x: γ list, hd x: bool より
  τ list = γ list, τ = bool
  故に γ = bool

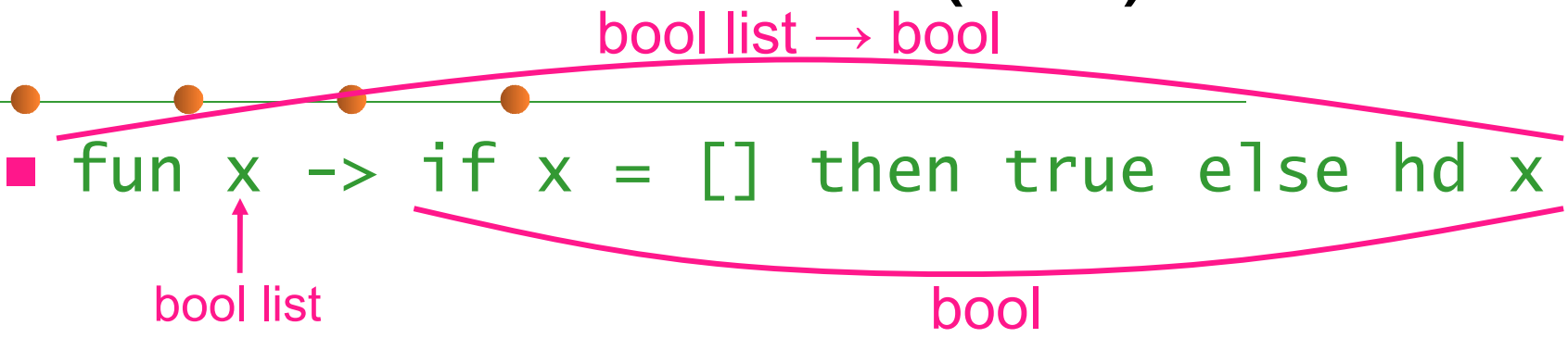
型付けの例 (6/7)



(hd : τ_{hd} := τ list → τ)

- { hd: τ_{hd}, x: α } の下で if ... の型を調べる
- if の条件節 x = [] より α = γ list
- then 節 true より (if ...) : bool
- hd: τ list → τ と
  x: γ list, hd x: bool より
  τ list = γ list, τ = bool
  故に γ = bool

型付けの例 (7/7)



(hd : τ_{hd} := τ list → τ)

- (fun x -> ...): α → β
- (if ...) : bool より β = bool
- x = γ list = τ list = bool list

- よって (fun x -> ...): bool list → bool

型推論の実装方針

- 式の構文の各要素
  (if 式、関数適用、let 式 etc.) について
  部分式と式全体の型に関する
  制約条件を解いていく
  - 条件が矛盾して制約解消失敗 → 型エラー
- 制約 =「型の間の等値関係」
- 制約解消 = unification

Unification とは

- 2つのパターンを一致させる代入を探すこと
  - 例1: `X, int ⇒ { X = int }`
  - 例2: `bool * X, Y * int ⇒ { X = int, Y = bool }`
  - 例3: `A → B, bool → C ⇒ { A = bool, B = C }`
    - 例3では `{ A = B = C = bool }` なども条件を満たす: 上のようにもっとも一般的なものを Most General Unifier (mgu) という
  - 例4: `A → B, bool ⇒` 失敗

Unification による型推論

- 例: `fun f → fun x → f x + f 1`
  - 部分式 e の型を τ(e) と書くことにすると
  - τ(`fun f…`) = α → β
  - τ(`fun x…`) = γ → δ = β [`fun f…` の返値]
  - τ(`f x + f 1`) = `int` = δ [`fun x…` の返値]
  - τ(`f x`) = `int`, τ(`f 1`) = `int`
  - τ(`f`) = α = γ → `int` = `int` → `int`

  - 結論: α = β = (`int` → `int`), δ = γ = `int`

    τ(`fun f…`) = (`int` → `int`) → `int` → `int`

型環境

- 自由変数の型に関する情報を保持
  - let 式や関数定義 (λ抽象) で必要となる
  - 例: let x = 5 in x + 3
    - x + 3 における型環境: { x : int }

型推論の実装における型の表現

- `type mltypes` (ファイル miniMLTyping.ml)
  - `TVar`: 型変数
    - フィールド `v` は変更可能 (mutable)
    - `TVar { id = n; v = TUnknown }`: 未定型変数
    - `TVar { id = _; v = (他の型) }`: `v` の型と同じ型

Unification の実装

- `unify` (ファイル miniMLTyping.ml)

- 今回は破壊的代入に基づいた
  アルゴリズムを用いる
  - `TVar { id = n; v = TUnknown }` と
    その他の型を unify する時に v のフィールドを
    直接もう1つの型で置換する
    - この `TVar` が別の `TVar` から参照されていれば
      自然に参照元の示す型も置換される
      → unification の結果が伝播される

型推論の実装

- `type_of_expr` (ファイル miniMLTyping.ml)
  - 式を受け取り
    部分式の型の間の制約を `unify` して
    全体の型を返す関数
  - 例1: `Plus, PairExp`
  - 例2: `IfExp` に対する実装
    - `new_type_variable ()` の使い方に注意
  - 例3: `LambdaExp`
    - 型環境の拡張 (generalize は後述)

型推論と多相型

- 型推論しても未決定の型変数が残ることがある
  - 例: fun x -> x からは

```
TArrow (
 TVar { id = 0; v = TUnknown },
 TVar { id = 1;
        v = TVar { id = 0; v = TUnknown }
      })
```

  といった型が推論される ('a → 'a に相当)
- 多相型とみなせそうだが…
  考えなければならない問題が2つある
  - 問題1: どの式を多相型にするか
  - 問題2: 多相型をどのように表現するか

問題1: どの式を多相型にするか

- ここでは ML 系言語にならい
  let 式で束縛された変数を多相型にする
  - ex. OK `let f = fun x -> x in (f 5, f true)`

    NG `(fun f -> (f 5, f true)) (fun x -> x)`

この下の様な例を扱うのは
一見簡単そうで
実は意外と面倒

問題2:
多相型をどのように表現するか

- 型スキーマとして表現する
  - 型スキーマ:
    - 型と
      その型中の未決定の型変数のうち
      多相的に扱うものの集合
      の組
- 例: hd の「型」: α list → α
  - これは使用時に α をどのような型に
    置き換えてもいいことを意味している
  - 型スキーマ ∀α. α list → α と表す

型スキーマの実装

- 型スキーマ $\forall\alpha_1 \ldots \alpha_n . \tau$ の表現
  - `type schema = id list * mltypes` (ファイル miniMLTyping.ml)
- 型から型スキーマへの変換
  - 関数 `generalize` (ファイル miniMLTyping.ml)
- 型環境の表現
  - `(ident * schema) list`

let 式の型推論の実装

- let x = e_1 in e_2 として
- まず e_1 の型推論を行う
  - その結果を τ とする
- 次に τ 中の未決定の型変数から型環境に含まれる型変数を除いたもので型スキーマ σ を作る
  - 関数 generalize
- 型環境を新たな束縛 [x : σ] で拡張して e_2 の型推論を行う

型環境に含まれる型変数は多相的に使えないことに注意

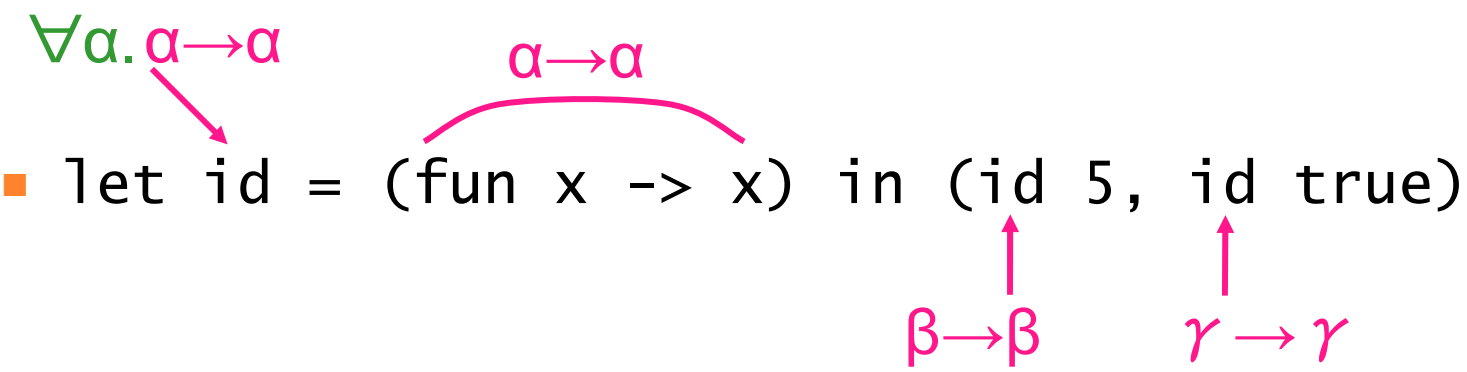
多相型を持つ変数の型付け

- まず型環境から変数に対応した型スキーマ $\forall\alpha_1 \ldots \alpha_n . \ \tau$ を取り出す
- 次に型変数 $\alpha_1 \ldots \alpha_n$ に対して新たに未決定の型変数 $\beta_1 \ldots \beta_n$ を作成し τ 中の $\alpha_1 \ldots \alpha_n$ を $\beta_1 \ldots \beta_n$ で置換する
  - これを τ' とする
- τ' を変数の型として返す
  - 実装: 関数 `instantiate` (ファイル miniMLTyping.ml)

多相型の型推論の例 (1/3)

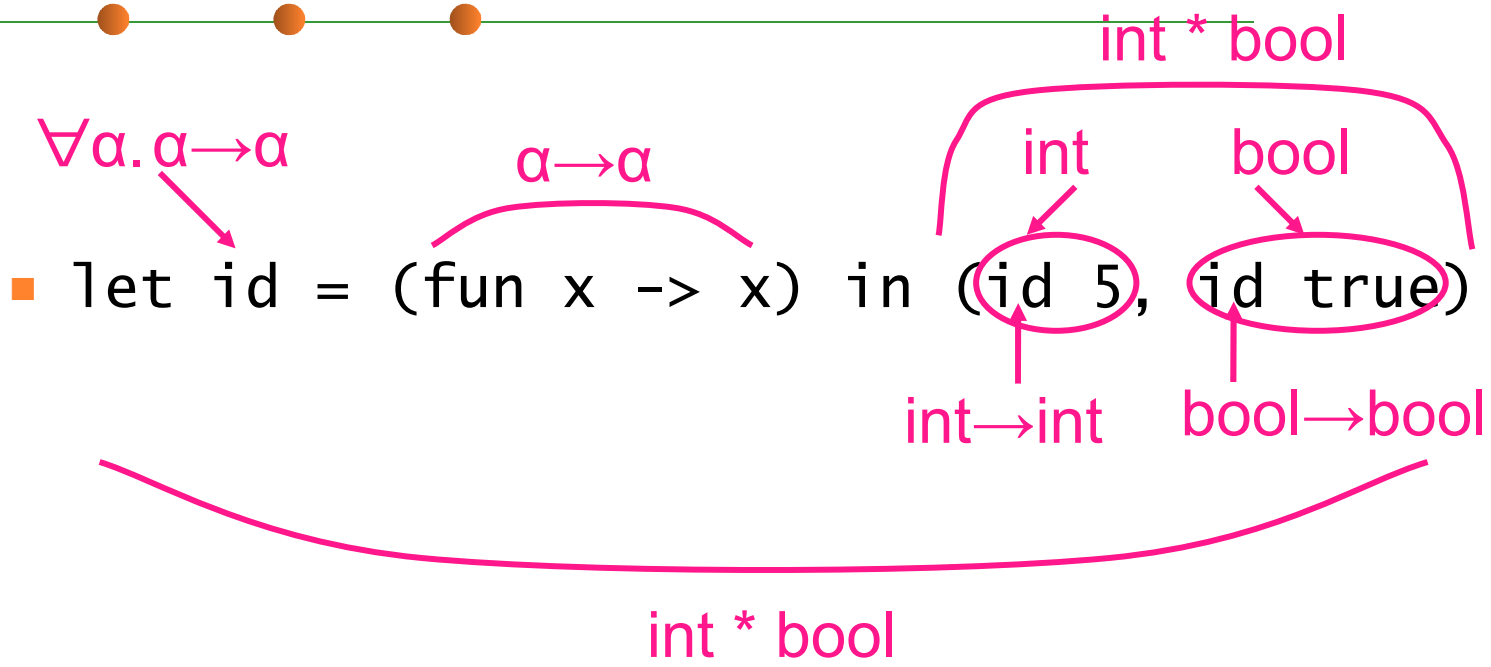
- `let id = (fun x -> x) in (id 5, id true)`
  - 型スキーマ (∀α.α→α) が `id` に束縛される

多相型の型推論の例 (2/3)



- 2つの id の出現が別の型変数に展開される

多相型の型推論の例 (3/3)



- id を多相的に使えている

第 7 回 課題

締め切り: 7/31 13:00

課題1 (必須)

- = (Equal) と ：： (ConsExp) に対する型推論処理を実装せよ
  - Equal の条件
    - (左辺の型) = (右辺の型)
    - 結果の型 = bool
  - Cons の条件
    - (結果型) = (右辺の型) = (左辺の型) list
- 関数適用の型推論を実装せよ

課題2 (必須)

- `let rec` 式
  `let rec f = e1 in e2`
  の型推論を実装せよ
  1. `[f : α]` を型環境に付け加えて e_1 を型推論
     - α は新たに作成した未決定の型変数
     - この段階では generalize しない (`Forall([], α)`)
  2. e_1 の型 τ と α を unify
  3. τ を generalize して σ にし
     `[f : σ]` を型環境に付け加えて e_2 を型推論

課題3 (Optional)

- match 式の型推論を実装せよ
  - match e_0 with p_1 -> e_1 | p_2 -> e_2 の形の式で
    何と何がマッチすればいいのかを考える
  - 関数 pattern_type を補助に使ってもよい
- function 式の型推論を実装せよ

課題4 (Optional)

- 一般の let 式、let rec 式の型推論を実装せよ
  - 基本は1引数・パターンなしの場合と同じ

課題5 (Optional)

- 第5回の eval 関数の実装と
  今回の型推論の実装を組み合わせて、
  型付き MiniML のインタプリタを作成せよ
  - プロンプト表示 → 式を受け取る
    → 評価結果と型を表示 → ... (繰り返し)
    - 型の表示には `print_mltypes` を使ってよい