

インタプリタ演習&型推論演習

工学部情報学科 3 回 橋田 遼太郎 入学年：2023 年、学籍番号：1029355810

提出日: 2025 年 7 月 28 日

提出期限: 2025 年 8 月 3 日

1

1.1 工夫した点

演算子優先順位の設計

文字列連結演算子 `^` の優先順位を適切に設定した。文字列インデックス `(.)` より優先度を低くし、図 1.1 のように `"hello".[0] ^ "world"` が正しく解釈されるようにした

```
# "hoge" ^ "hoge" ではなく "hoge" ^ "hoge"
val it : string = "hworld"
```

図 1.1

文字インデックスの字句解析

`.` と `[` を分離せず、`.[` として認識することで、`s.[n]` 構文の解析を簡潔にした。これにより `s.field[index]` のような他の用途との衝突を回避できるようにした。

```
1 | ".[" { DOT_LBRACKET }
2 | "]" { RBRACKET }
```

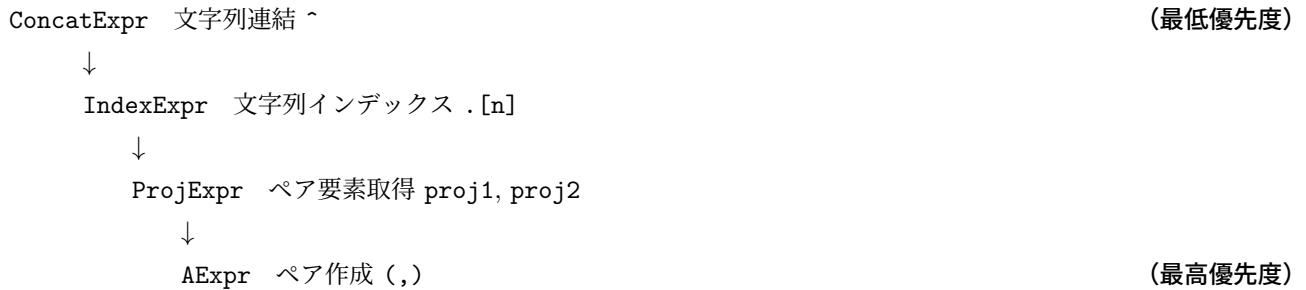
1.2 テスト結果

```
# let s = "hello";
val s : string = "hello"
# s.[1];
val it : string = "e"
# s^s;
val it : string = "hellohello"
# print_string s;
helloval it : string = "hello"
```

2 ペア型の値とそれに対する操作を実装しなさい。型推論までサポートすること。

2.1 工夫した点

演算子優先順位の設計



ペア操作を他の演算より高い優先度に設定した。これにより、`proj1 p ^ "suffix"`が`(proj1 p) ^ "suffix"`として正しく解析される。

```
# let p = ("hoge", "world");;
val p : (string * string) = ("hoge", "world")
# proj1 p ^ "hello";;
val it : string = "hogehello"
```

図 2.1

制約ベース型推論

```
1  let rec ty_exp tyenv = function
2  | StrConcatExp (exp1, exp2) ->
3      let (s1, ty1) = ty_exp tyenv exp1 in
4      let (s2, ty2) = ty_exp tyenv exp2 in
5      let eqs = [(ty1, TyString); (ty2, TyString)] in (* 制約生成 *)
6      let s3 = unify eqs in (* 制約解決 *)
7      (compose_subst s3 (compose_subst s2 s1), TyString)
```

型推論を制約生成と制約解決の2段階に分離した。型制約を明示的に生成してから統一することで、複雑な型推論も系統的に処理可能。

2.2 テスト結果

```
# let p = ("hello",30);;
val p : (string * int) = ("hello", 30)
# proj1 p;;
val it : string = "hello"
# proj2 p;;
val it : int = 30
```