レコードとハッシュテーブルの暗黙な相互運用を可能にする型推論とコンパイル手法

梅木孝輔　京都大学工学部情報学科
関山太朗　国立情報学研究所
五十嵐淳　京都大学大学院情報学研究科

動機

- 静的なデータ構造であるレコードと動的なデータ構造であるハッシュテーブルの双方の長所を生かしたい

```
/*レコードの宣言*/
let r_person = {Name="Joe"; Age=21};;

/*ハッシュテーブルの宣言*/
let h_person = create;;
add h_person "Name" "Alice";;

/*レコードとしてのアクセス*/
let get_name = λx. x.Name in
get_name h_person;;

/*ハッシュテーブルとしてのアクセス*/
let get_name = λx. find x "Name" in
get_name r_person;;
```

ラベルはコンパイル時に決定＝仕様が確定
- 効率の良いアクセス
- 頑健なプログラム

ラベルは実行時に決定＝仕様が未定
- 柔軟なプロトタイピング

概観

- レコードとハッシュテーブルの暗黙な相互運用を行える体系の提案

レコード を ハッシュテーブル としてアクセス
ハッシュテーブル を レコード としてアクセス
を型注釈無で実現

- 動的なデータ変換は伴わない

研究内容

- 右を参照

将来の課題

- 型付け・型推論・コンパイルの正しさの証明
- 参照実装

研究内容

レコードとハッシュテーブルを持つ形式的体系の定義

① $\lambda^{let,.}$ ⟹② $\Delta^{let,.}$ ⟹③ ④ $\lambda^{let,[\,]}$

中間言語

① レコードとハッシュテーブルを備えたラムダ計算

$$
\begin{aligned}
e ::= {} & x \mid \lambda x.e \mid e_1\ e_2 \\
& \mid \text{let } x = v_1 \text{ in } e_2 \\
& \mid \{l_1 = e_1; \ldots l_n = e_n\} \\
& \mid e.l \mid e_1.l \leftarrow e_2 \\
& \mid \text{create} \mid \text{find } e_1\ e_2 \\
& \mid \text{add } e_1\ e_2\ e_3 \ \ldots
\end{aligned}
$$

② ハッシュテーブルに対し拡張されたカインドを用いた型付け・型推論

- Ohoriの多相レコード[TOPLAS'95]カインド付け
- レコードカインド$\{\{l_1 : \tau_1\}\}$ -> 少なくとも$l_1 : \tau_1$を持つようなレコード型に付く

$$\frac{\mathcal{K}(t) = \{\{l_1 : \tau_1; \ldots l_n : \tau_n; \ldots\}\}}{\mathcal{K} \vdash \{l_1 : \tau_1; \ldots l_n : \tau_n; \ldots\} :: \{\{l_1 : \tau_1; \ldots l_n : \tau_n\}\}}$$

```
r1:{Name:string; Age:int}::{ Name:string }
r2:{Name:string;
    Address:string}::{ Name:string }

/*レコードの宣言*/
let r1 = {Name="Joe"; Age=21};;
let r2 = {Name="Bob"; Address="Kyoto"};;

/*レコードとしてのアクセス*/
let get_name = λx. x.Name in
get_name r1;;
get_name r2;;
x:t1::{ Name: t2}
```

拡張

- 全てのレコード型とハッシュテーブル型を包摂する { } の導入

```
r:{Name:string; Age:int}::{ }

/*レコードの宣言*/
let r = {Name="Joe"; Age=21};;

/*ハッシュテーブルとしてのアクセス*/
let get_name = λx. find x "Name" in get_name r;;
x:t1::{ }
```

すべてのレコード・ハッシュテーブルに付けることができる

- ハッシュテーブル型は任意のレコードカインドを取る

```
h:hashtbl::{ Name:string; Age:int }
h:hashtbl::{ Age:int }
h:hashtbl::{ Name:string }

/*ハッシュテーブルの宣言*/
let h = create;;
add h "Name" "Alice";;
x:t1::{Name:t2}

/*レコードとしてのアクセス*/
let get_name = λx. x.Name in get_name h;;
```

③ レコードアクセスのコンパイル

- 引数がハッシュテーブルかレコードか動的ディスパッチするコードへのコンパイル

```
/*レコードとしてのアクセス(コンパイル後)*/
let get_name = λI.λt1.λt2.λx.
          btlz I x[I] x["Name"]:t1
```

次の添字Iが0未満か0以上かで動的に分岐

```
/*ハッシュテーブルの宣言*/
let h = create;;
add h "Name" "Alice";;

/*レコードの宣言*/
let r = {Name="Joe"; Age=21};;
/*レコードとしてのアクセス(コンパイル後)*/
let get_name = λI.λt1.λt2.λx.
          btlz I x[I] x["Name"]:t1
in get_name hashtbl string -1 h
in get_name {Name:string; Age:int} string 1 r
```

④ 配列とハッシュテーブルを備えたラムダ計算

- ハッシュテーブルの値の動的な型チェック
- ハッシュテーブルアクセスのためのデータ構造の低水準実装方針
  - Lookup/Update関数 = レコードをハッシュテーブルアクセスする為の疑似的なハッシュ関数を表現