コンパイラ演習
第 8 回

2007/11/22

小酒井 隆広　前田 俊行
山下 諒蔵　佐藤 春旗
大山 恵弘　佐藤 秀明
住井 英二郎

今日の内容

- バリアント / レコード
  - 表現方法
  - 型付け
- パターンマッチ
  - 型付け
  - if 式への変換

バリアント / レコード

バリアントのメモリレイアウト

- 先頭にタグを追加したタプルのように配置
  - タグ名は整数にエンコード
    - 異なるバリアント型のタグには同じ整数値を使用してもよい
  - 要素数は可変

```
type t =
  A of string * int
| B
| C of string
```

A → 0
B → 1
C → 2

```
A("a",12)
B
C("c")
```

```
(0,"a",12)
(1)
(2,"c")
```

レコードのメモリレイアウト

- ラベル名でソートしたタプルのように配置

```
type s =
{ foo : string;
  bar : int;
  baz : float }
```

```
{ foo = "";
  bar = 12;
  baz = 1.23 }
```

```
(12, 1.23, "")
```

中間コード上での表現

- 新たにプリミティブを追加
  - バリアント: 生成、タグ取得、要素取得
  - レコード: 生成、要素取得
- または、既存のプリミティブに吸収してもよい
- 実装方法は各自の判断に任せます

バリアントの型付け

- タグ名と型情報との関連を管理
  - タグ名から型が一意に決まるように
    型付け規則を決めると楽
    - つまり、複数のバリアント型で同じタグ名が
      使われていたら型エラーとする
  - なお、一意に決まらないようなシステムも可能だが…

```
type t =
  A of string * int
| B
| C of string
```

```
TagEnv(A) =
  (t, [string; int])
TagEnv(B) =
  (t, [])
TagEnv(C) =
  (t, [string])
```

レコードの型付け

- ラベル名と型情報との関連を管理
  - ラベル名から型が一意に決まるように
    型付け規則を決めると楽
    - つまり、複数のレコード型で同じラベル名が
      使われていたら型エラーとする
  - なお、一意に決まらないようなシステムも可能だが…

```
type s =
{ foo : string;
  bar : int;
  baz : float }
```

```
  LabelEnv(bar)
= LabelEnv(baz)
= LabelEnv(foo)
= (s, [bar → int;
       baz → float;
       foo → string])
```

パターンマッチ

パターン式の導入

- 基本的な定義

p ::=	（パターン式）
| _	（ワイルドカードパターン）
| c	（定数パターン）
| x	（変数パターン）
| $(p_1, \ldots, p_n)$	（タプルパターン）
| $C\ p$	（バリアントパターン）
| $\{l_1 = p_1; \ldots; l_n = p_n\}$	（レコードパターン）

パターン式の型付け

$$\Gamma \vdash p : \tau, (B)$$

Γ：自由変数の型環境
p：パターン式
τ：p の型
B：p の中で束縛される変数の型環境

- 読み方：
  型環境 Γ のもとでパターン p は型 τ に
  型付けされ、p 中で束縛される変数の型環境は
  B となる

パターン式の型付け規則（抜粋）

（変数パターン）

$$\Gamma \vdash x : \tau, (x : \tau)$$

パターン式で束縛
された変数の型環境

（タプルパターン）

$$\frac{\Gamma \vdash p_1 : \tau_1, (B_1) \quad \ldots \quad \Gamma \vdash p_n : \tau_n, (B_n) \qquad \text{for all } i \neq j,\ \mathrm{dom}(B_i) \cap \mathrm{dom}(B_j) = 0}{\Gamma \vdash (p_1, \ldots, p_n) : \tau_1 \times \ldots \times \tau_n, (B_1, \ldots, B_n)}$$

タプルの各要素で束縛された変数の
名前が重複していないことを確認

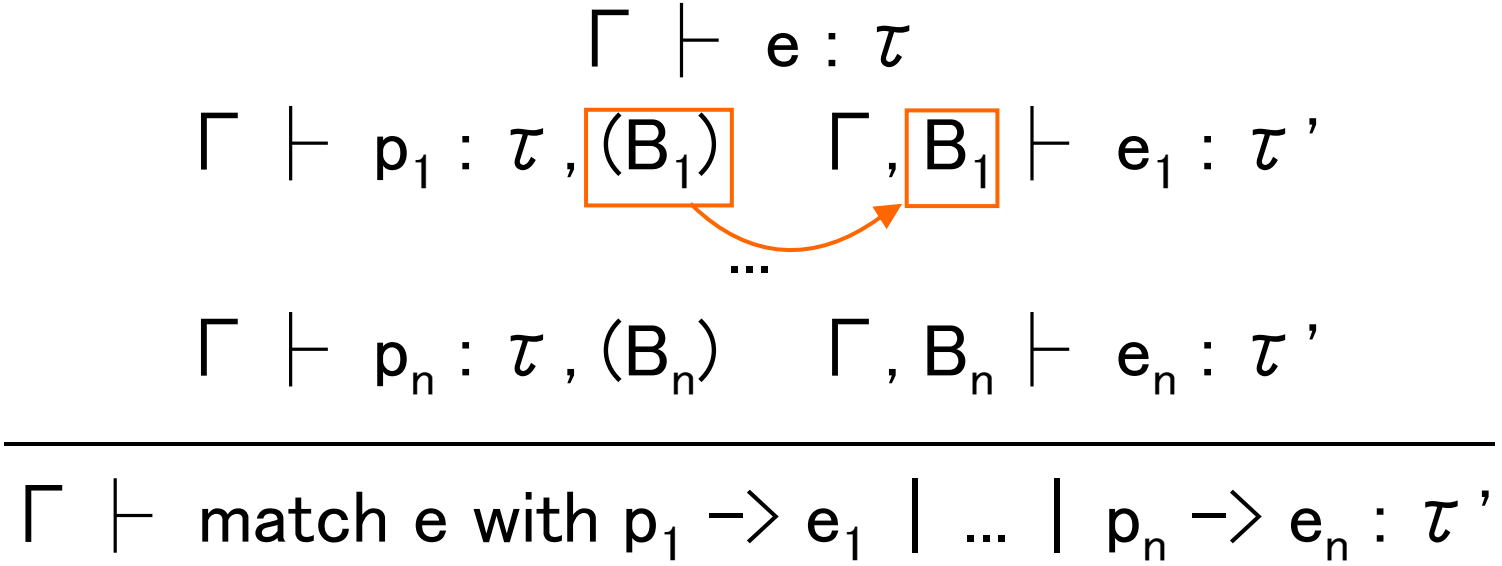
match 式の導入

- 基本的な定義

```
e ::=                          （式）
  | …
  | match e with               （パターンマッチ式）
        p1 -> e1
      | …
      | pn -> en
```

match 式の型付け規則

- パターン式が束縛した変数の型環境を利用



パターンマッチのナイーブな実装方法

- if 式に変換する

```
match e with
   p1 -> e1
| …
| pn -> en
```

```
let v = e in
if test (p1,v) then
   bind (p1,v)
   e1
else
   …
else if test (pn,v) then
   bind (pn,v)
   en
```

v が p_1 にマッチするかどうかを表す式に変換

p_1 の中の変数パターンを束縛する式に変換

パターンマッチの
ナイーブな実装方法の例

```
match e with
   (1,x) -> e1
 | (y,2) -> e2
```

```
let v = e in
if (let (t,_) = v in
    t = 1) then
  let (_,x) = v in
  e1
else
if (let (_,t) = v in
    t = 2) then
  let (y,_) = v in
  e2
```

パターンマッチのコンパイルに関する参考文献

- Caml Light でのパターンマッチ実装法
  - Xavier Leroy, “The ZINC experiment: an economical implementation of the ML language”
    - http://pauillac.inria.fr/~xleroy/publi/ZINC.ps.gz
- O’Caml が採用したパターンマッチ最適化の手法
  - Fabrice Le Fessant, Luc Maranget, “Optimizing Pattern-Matching”
    - http://pauillac.inria.fr/~maranget/papers/opt-pat.ps.gz
- パターンマッチ最適化の heuristics
  - Kevin Scott and Norman Ramsey, “When Do Match-compilation Heuristics Matter?”
    - http://www.cs.virginia.edu/~techrep/CS-2000-13.pdf

共通課題

- 次の match 式を if 式に変換せよ
  - 「最良」と思われる変換結果を提出せよ
  - 「最良」の定義は各自で設定すること

```
match e with
   (_, 1, 4) -> e1
| (_, 1, 5) -> e2
| (1, 2, 3) -> e3
| (3, 2, 3) -> e4
| (1, 1, 2) -> e5
| (1, 1, x) -> e6
| (y, _, _) -> e7
```

コンパイラ係用選択課題

- バリアントとパターンマッチを、
  自作コンパイラまたは MinCaml に実装せよ
  - パターンマッチでは、少なくとも以下のパターンを扱えるようにすること
    - ワイルドカードパターン
    - 定数パターン
    - 変数パターン
    - バリアントパターン
  - パターンに 1 つもマッチしなかったときは
    どうすれば良いか考えて実装せよ

課題の提出先と締め切り

- 提出先: compiler-report-2007@yl.is.s.u-tokyo.ac.jp
- 締め切り: 2 週間後 (12/6) の午後 1 時
  - コンパイラ係用課題の締め切り: 2008年 2月 29日
- Subject: **Report␣␣8** 〈学籍番号〉

半角スペース 1 個ずつ

  - 例: **Report 8 71099**
- 本文にも氏名と学籍番号を明記のこと

◆ 質問は compiler-query-2007@yl.is.s.u-tokyo.ac.jp まで