

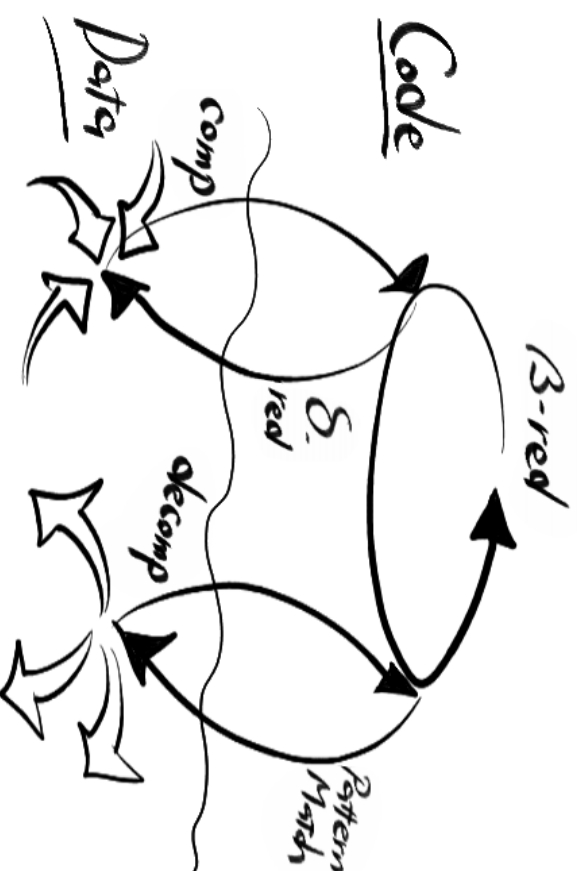
汎用的関数型言語における系統的な
外部リソース操作の原理と実装

大和谷 潔: 学生番号 010119

目的

- DBレコードやオブジェクトなど種々のデータモデルにもとづくデータを関数型言語で扱いたい。
- このようなデータを提供する他の言語環境向けの既存のライブラリを関数型言語で使いたい。
- パターンマッチは関数型言語の魅力の一つ。このような外部データを対象に、組み込みデータと同様のパターンマッチがおこなえるようにしたい。

コードとデータ



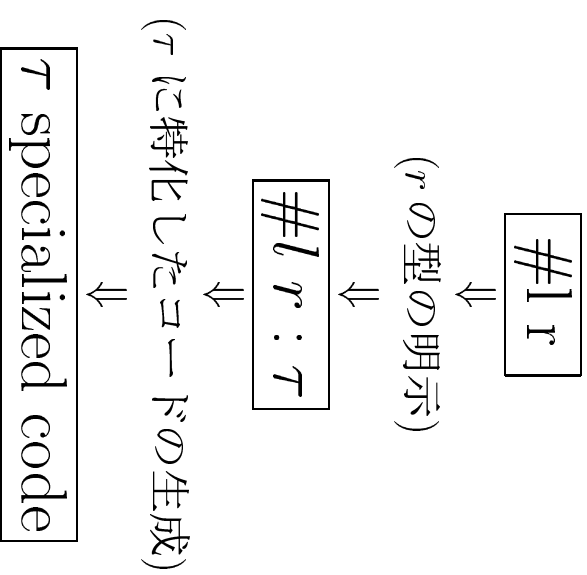
コードとデータは、簡約 (プリミティブ呼出) とパターンマッチのみで連絡。

⇒ ここで、データモデルごとには、簡約とパターンマッチの機構を用意する。

⇒ これにより、コードは異なるデータモデルの外部データを統一的に扱うことができる。

方針

- δ 簡約時には、外部データへの参照のみを取り込む。(一括コピーはおこなわない。)
- (パターンマッチはファイルド取出し式 $\#l r$ に変換される。) ファイルド取出し式をつぎのように変換する。



概要

以上の方針を、つぎのように実現する。

外部型 外部データを表現する型を導入する。

型システム 外部型を組み込んだ型システムを設計する。

コンパイラ 型システムによる式の型づけを利用してプログラムをコンパイルする。

ランタイム 抽象機械コードと、ネイティブコードとのインタフェイスを実装する。

外部型

```
externaltype  $T$  as { $l_1 : \tau_1, \dots, l_n : \tau_n$ } importing "dom : name"
```

- dom : $name$ が指す外部データの型を T と宣言する。 dom をドメイン、 $name$ を外部名と呼ぶ。
- T の値はファイル $l_1 : \tau_1, \dots, l_n : \tau_n$ をもつデータとして扱うことができる。ファイルドの取出し方法はドメイン依存。

```
externaltype WebBrowser as {url : string, hwnd : int}  
importing "com : SHDocVwCtl1.WebBrowser"
```

- COMオブジェクトを表す型を `WebBrowser` 型と宣言。
- `WebBrowser` 型の値は `url` と `hwnd` の二つのファイルドを持つ。
- ファイルドの取出し方法はCOMのルールにしたがう。

型システム

- kinded type system ([ohori95]) をもとに、外部型を組み込んだ型システムを設計する。

目的

- 式に型を与えることで、コンパイラの手がかりとする。
- パターンマッチにおいて、外部型の値と組み込みのレコードと同様に扱えることを保証する。
- 外部データのみを引数に期待する外部関数に、組み込みのレコードやその他の外部型の値を渡さないことを保証する。

kinded type system

kind 型の上位レベルの概念。

値の集合 \Rightarrow 型

型の集合 \Rightarrow カインド

カインドを与えられた型変数

$$\alpha :: \{l_1 : \tau_1, \dots, l_n : \tau_n\}$$

l_1, \dots, l_n までのフィールドをもつ型のみ、型変数 α に代入可能であることを意味する。

例

```
val getName = fn x => #Name x
```

getName は、フィールド Name をもつあらゆる型を引数にとることができる。

```
getName :  $\forall \alpha :: \{\}$ .  $\forall \beta :: \{\text{Name} : \alpha\}$ .  $\beta \rightarrow \alpha$ 
```

型システムが満たすべき要件

以下の要件を満たす型システムを設計することで目的を達成できる。

$$\left(\begin{array}{l} \text{external type } T \text{ as } \{l_1 : \tau_1, \dots, l_n : \tau_n\} \\ \text{external type } T' \text{ as } \{l_1 : \tau_1, \dots, l_n : \tau_n\} \\ v : T \quad v' : T' \end{array} \right) \text{ とする。}$$

- $f : T \rightarrow \tau$ である場合、 f は認めるが、 $f \{l_1 = e_1, \dots, l_n = e_n\}$ は認めない。
- $g : \forall \alpha :: \{l_1 : \tau_1, \dots, l_n : \tau_n\}. \alpha \rightarrow \tau$ である場合、 g は認める。
- $h : \{l_1 : \tau_1, \dots, l_n : \tau_n\} \rightarrow \tau$ である場合、 h は認めない。
- T と T' とは互いに相容れない型とする。たとえば、`if e then v else v'` は認めない。

⇒ このような要件を満たす型システムを設計する。

コンパイル手順

ソースプログラム



明示的に型づけされた式



参照ファイルドの判定



ファイルド選択関数を受け渡す式



ドメイン依存コード

コンパイル-1

ソースプログラムの式 \Rightarrow 明示的に型づけされた式

ファイル抽出し式 ファイルドを保持する式の型を明示する。

$$\frac{\mathcal{T} \triangleright r : \tau}{\mathcal{T} \vdash \#lbl\ r \Rightarrow \#lbl\ r : \tau}$$

変数式 polymorphic な変数 (=関数) を使用する個所において、型変数に代入される型を文脈から判定し、明示する。

$$\frac{\mathcal{T} = \mathcal{T}' \{x : \forall \alpha_1 :: k_1 \dots \alpha_n :: k_n. \tau\} \triangleright x : [\tau_n / \alpha_n] \dots [\tau_1 / \alpha_1] \tau}{\mathcal{T} \vdash x \Rightarrow x\ \tau_1 \dots \tau_n}$$

val getUrl = fn r \Rightarrow #Url r

\Rightarrow val getUrl : $\forall \alpha :: \{\}\}. \forall \beta :: \{\text{Url} : \alpha\}. \beta \rightarrow \alpha = \text{fn } r \Rightarrow \#Url\ r : \beta$

getUrl (wb : WebBrowser)

($\beta \mapsto \text{WebBrowser}, \text{WebBrowser}$ は $\text{Url} : \text{string}$ をもつ $\Rightarrow \alpha \mapsto \text{string}$)

\Rightarrow (getUrl string WebBrowser) wb

コンパイル-2

明示的に型づけされた式 \Rightarrow 参照フレイルドの判定

式に与えられた型から、式が参照するフレイルドのリスト

$[fld(lbl_1, \tau_1), \dots, fld(lbl_n, \tau_n)]$ を判定する。

($fld(lbl, \tau)$ は、型 τ に含まれるフレイルド lbl を指す。)

```
val getUrl :  $\forall \alpha :: \{\}\}. \forall \beta :: \{\text{Url} : \alpha\}. \beta \rightarrow \alpha = \text{fn } r \Rightarrow \#\text{Url } r : \beta$ 
```

```
 $\Rightarrow$  getUrl : [ $fld(\text{Url}, \beta)$ ]
```

```
(getUrl string WebBrowser) wb
```

```
 $\Rightarrow$  getUrl : [ $fld(\text{Url}, \text{WebBrowser})$ ] wb
```

```
( $\beta \mapsto \text{WebBrowser}$  より)
```

コンパイル-3

参照フイルド ⇒ フイルド選択関数を受け渡す式

ある polymorphic な関数 f が、 $fld(lbl, \alpha)$ を参照している。

⇒ α に与えられる型を f は決定できないので、

f は、このフイルドを取出す方法が分からない。

一方、 f を呼び出す式 e は、 α に代入される型 τ を決定でき、 τ から lbl フイルドを取出す方法を決定できる。

⇒ $fld(lbl, \tau)$ を取出すコード $sel(lbl, \tau)$ を、関数の引数として e から f へ受け渡す。
 f は引数で渡されるコードを用いてフイルドを取出す。

```
getUrl : [fld(Url,  $\beta$ )]
```

```
⇒ val getUrl :  $\forall \alpha. \forall \beta. sel(Url, \beta) \rightarrow \beta \rightarrow \alpha = \text{fn } f \Rightarrow \text{fn } r \Rightarrow f\ r$ 
```

```
getUrl : [fld(Url, WebBrowser)] wb
```

```
⇒ getUrl sel(Url, WebBrowser) wb
```

コンパイル-4

ファイルド選択関数を受け渡す式 ⇒ ドメイン依存コード

$sel(lbl, \tau)$ を、 τ のドメインに依存したコードにコンパイルする。

`getUrl sel(url, WebBrowser) wb`

⇒ `getUrl (fn o ⇒ o → getUrl()) wb`

- `com` ドメインは `WebBrowser` 型の `Url` ファイルド取出しを、`COM` オブジェクトの `getUrl` メソッド呼び出しにマッピングしている。
- 下線を施した部分は、`C++` で記述した同様のコードに相当する。

フイルド参照式の変換例

`val getUrl = fn r => #url r`

(型の明示)

$\Rightarrow \text{val getUrl} : \forall \alpha :: \{\!\!\{ \}\!\!\}. \forall \beta :: \{\text{url} : \alpha\}. \beta \rightarrow \alpha = \text{fn } r : \beta :: \{\text{url} : \alpha\} \Rightarrow \#url\ r : \beta$

(フイルド引数の追加)

$\Rightarrow \text{val getUrl} : \forall \alpha. \forall \beta. \text{sel}(\text{url}, \beta) \rightarrow \beta \rightarrow \alpha = \text{fn } f : \text{sel}(\text{url}, \beta) \Rightarrow \text{fn } r \Rightarrow f\ r$

ファイルを参照する polymorphic 関数を使用する式の変換例

`getUrl (wb : WebBrowser)`

(型の明示)

\Rightarrow `getUrl string WebBrowser) wb`

(ファイルド引数の追加)

\Rightarrow `getUrl sel(URL, WebBrowser) wb`

(ドメイン依存コード生成)

\Rightarrow `getUrl (fn o \Rightarrow o \rightarrow getUrl()) wb`

現在の状況

型付きラムダ式のコンパイラを実装。

型推論 / record calculus polymorphic-record [ohori95]

パターンマッチコンパイラ term-decomposition

操作意味論 / ランタイム zinc abstract machine (Caml-light)

今後の予定

- 外部型を導入した型システムの考察
- 型のマッピングの検討 (例: `char*` → `string?` / `char list?`)
- コンパイラおよびランタイムの設計 (とくに、フレーム依存コードを生成するモジュールをコンパイラ本体に組み込む方法)
- 実装
- 評価