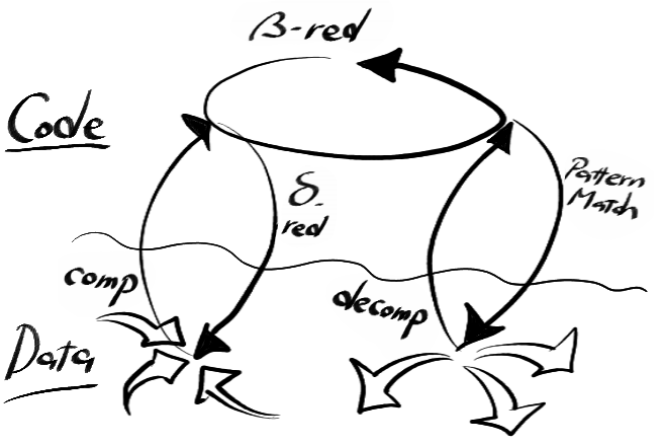
1　背景および目的

実用的なプログラミング言語は、豊富なライブラリを用意して、データベースやグラフィックなど外部リソースの操作手段をプログラマに提供している。これらのライブラリを関数型言語プログラムから扱うことができれば、関数型言語の実用性が増す。しかし、これらのライブラリが公開するデータモデルは、必ずしも関数型言語のそれとは一致しない。本研究は、外部ライブラリの提供するデータを実行時に直接操作しながらも、ソースコード上で関数型言語の組み込みデータと同様に扱う方法を探る。

2　着想

関数型言語プログラムは、コード (式) とデータ (値) とを操作しながら実行を進める。

δ 簡約によってデータが合成 (composition) され、コードの世界へ送り込まれる。パターンマッチによってデータが分解 (decomposition) され、一部を除いてコードの世界から解放される。



コードの世界とデータの世界は、δ 簡約とパターンマッチによってのみ連絡する。言い換えれば、データモデルに依存するのは δ 簡約とパターンマッチのみである。したがって、各種のデータモデルに対応して δ 簡約およびパターンマッチを実現する機構を用意すれば、コードは種々の外部リソースを自然な形で操作することができる。

外部データを関数型言語の世界に取り込む方法には、つぎの二通りがある。

- δ 簡約の際に、外部データのフィールド値をレコードやリストに一度にコピーする方法。データモデルへの依存性が δ 簡約のみに限定され、パターンマッチは組み込みのデータ型のみを対象にすればよい利点がある。しかし、コピー後に元データの状態が変化してもコピー先には反映されない欠点がある。
- δ 簡約時には外部データへの参照のみを受け取り、パターンマッチ時に外部関数を通してそのフィールド値を得る方法。パターンマッチの実装は、データモデルに依存する。

本研究は後者の方法をとる。パターンマッチはフィールド取出し演算と比較演算を組み合わせて実現される。このうち、比較演算は組み込みの基本型を対象とする。したがって、フィールド取出し演算に関してのデータモデル依存性を解決すればよい。

3 概要

フィールド取出し式 $\#l\ r$ を、r の型を明示した $\#l\ r : \tau$ に変換し、最終的に、τ に依存したコードにコンパイルする。とくに τ が外部ライブラリの提供する型ならば、そのライブラリのデータモデルに特化したコードにコンパイルする。

$$\#l\ r \quad \Rightarrow \quad \#l\ r : \tau \quad \Rightarrow \quad \tau\text{に特化したコード}$$

ところが、単純な型システムのもとではフィールド取出し式を型づけすることができない。そこで、[ohori95] で示された polymorphic record calculus を用いる。

3.1 外部型

外部リソースを表現する型をつぎのように宣言する。

$$\texttt{externaltype}\ T\ \texttt{as}\ \{l_1 : \tau_1, \ldots, l_n : \tau_n\}\ \texttt{importing}\ ''dom : name''$$

これは、$dom : name$ で指定される外部リソースの型を T と表すことを宣言する。dom をドメイン、$name$ を外部名と呼ぶ。$\{l_1 : \tau_1, \ldots, l_n : \tau_n\}$ は、型 T の値がフィールド $l_1 \ldots l_n$ を持つことも示している。

$\texttt{externaltype WebBrowser as}\ \{\texttt{Url} : \texttt{string}, \texttt{hWnd} : \texttt{int}\}\ \texttt{importing}\ ''\texttt{com} : \texttt{SHDocVwCtl.WebBrowser}''$

3.2 型システム

[ohori95] に示された型システムをもとに、外部型に関して以下の要件を満たす型システムを設計する。(ただし、$\texttt{externaltype}\ T\ \texttt{as}\ \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$ および $\texttt{externaltype}\ T'\ \texttt{as}\ \{l_1 : \tau_1, \ldots, l_n : \tau_n\}$ と宣言され、v を T 型の値とし、v' を T' 型の値とする。) ($\alpha :: \{\!\{l_1 : \tau_1, \ldots, l_n : \tau_n\}\!\}$ は、$l_1, \ldots, l_n$ 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。)

- $f : T \rightarrow \tau$ である場合、$f\ v$ は認めるが、$f\ \{l_1 = e_1, \ldots, l_n = e_n\}$ は認めない。
- $g : \forall \alpha :: \{\!\{l_1 : \tau_1, \ldots, l_n : \tau_n\}\!\}.\alpha \rightarrow \tau$ である場合、$g\ v$ は認める。
- $h : \{l_1 : \tau_1, \ldots, l_n : \tau_n\} \rightarrow \tau$ である場合、$h\ v$ 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、$\texttt{if}\ e\ \texttt{then}\ v\ \texttt{else}\ v'$ は認めない。
- T 型の式とパターン $\{l_1 = p_1 : \tau_1, \ldots, l_n = p_n : \tau_n\}$ とは照合できない。
- T 型の式はパターン $\{l_1 = p_1 : \tau_1, \ldots, l_n = p_n : \tau_n, \ldots\}$ とは照合できる。

3.3 ドメイン依存コードの生成

まず、ソースプログラムの式を、明示的に型づけされた式に変換する。

- フィールド取出し式 $\#lbl\ r$ を、r の型を明示した $\#lbl\ r : \tau$ に変換する。

- 変数 x が型 $\forall\alpha_1 :: k_1 \ldots \alpha_n :: k_n.\tau$ として束縛されている文脈のもとで、x を型 $[\tau_n/\alpha_n]\ldots[\tau_1/\alpha_1]\tau$ の式として参照している変数式 x を、x に型を与える式 $x\ \tau_1\ \ldots\ \tau_n$ に変換する。

こうして明示された型から、式が参照するフィールドの集合 $\{fld(lbl_1,\tau_1),\ldots,fld(lbl_n,\tau_n)\}$ が判定できる。($fld(lbl,\tau)$ は、型 τ に含まれるフィールド lbl を指す。)

そして、式が参照するフィールドのそれぞれについて、その取出しを実行するコードを受け渡すように変換する。とくに τ が外部型である場合、そのドメインに応じたコードを受け渡す。

$$
\begin{array}{rcl}
 & & \texttt{val getUrl} \ = \ \texttt{fn}\ r \ \Rightarrow \ \#\texttt{Url}\ r \\
\text{(型の明示)} & \Rightarrow & \texttt{val getUrl} : \forall\alpha :: \{\!\{\}\!\}.\forall\beta :: \{\!\{\texttt{Url} : \alpha\}\!\}.\beta \to \alpha \ = \ \texttt{fn}\ r : \beta :: \{\!\{\texttt{Url} : \alpha\}\!\} \Rightarrow \#\texttt{Url}\ r : \beta \\
\text{(フィールド引数の追加)} & \Rightarrow & \texttt{val getUrl} : \forall\alpha.\forall\beta.fld(\texttt{Url},\beta) \to \beta \to \alpha \ = \ \texttt{fn}\ f : fld(\texttt{Url},\beta)\ r \ \Rightarrow \ f\ r
\end{array}
$$

$$
\begin{array}{rcl}
 & & \texttt{getUrl}\ wb \\
\text{(型の明示)} & \Rightarrow & (\texttt{getUrl string WebBrowser})\ wb \\
\text{(フィールド引数の追加)} & \Rightarrow & \texttt{getUrl}\ fld(\texttt{Url},\texttt{WebBrowser})\ wb \\
\text{(ドメイン依存コード生成)} & \Rightarrow & \texttt{getUrl}\ (\texttt{fn}\ o \Rightarrow \underline{o \to \texttt{getUrl}()})\ wb
\end{array}
$$

wb は、前に例として示した外部型 `WebBrowser` の変数であるとする。そして、`com` ドメインは `WebBrowser` 型の `Url` フィールドを 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。

4 現在の状況

型付きラムダ式のコンパイラを実装した。コンパイラを構成する主なモジュールは、以下の研究成果を参考としている。

型推論/record calculus polymorphic-record[ohori95]

パターンマッチコンパイル term-decomposition

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

5 今後の予定

- 外部型を導入した型システムの考察
- コンパイラの設計 (とくに、ドメイン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)
- 外部データとガベージコレクションの関係の考察
- 実装
- 評価