7F－1

Haskell言語上のClassおよびMonadを用いたデータベース操作インターフェイスの実装

喜多村晶子　竹島由里子　市川哲彦　藤代一成　佐藤浩史

お茶の水女子大学　理学部

1　はじめに

本研究は、純粋な関数型算譜言語(pure FP)とデータベース(DB)の操作体系との統合を目的としている。

関数型データベース算譜言語(DBPL)の研究・開発が行われる一方で、pure FPは種々の参照透過な入出力機構を備えてきた[1] [2]。従って、Nikhil ([3])に指摘されたような、順次実行制御機構の欠落に起因する更新操作記述の困難さは解消可能である。

関数型のDBPLではDBは記号束縛の環境で与えられ、更新は記号束縛の変更／生成、または記憶域に束縛された変数への代入により行われる。一方、pure FPでは実行時にトップレベルの束縛環境変更はできないため、なんらかの形で変数操作に対応する操作体系を持ち込む必要がある。

本研究では、関数型算譜言語Haskell [4]上に、モナド(Monad) [2]による順次実行制御と、多重定義関数によるDB操作体系の導入を行った。多重定義はクラス(class)機構によって制御されているが、言語仕様の若干の変更が必要であったため、Glasgow Haskell Compiler [5]に手を加える形で、これを実現した。簡単な操作例を図に示す。これは“university” DB中の学生データを更新する例であり、findStudentは名前で学生データを検索するためのユーザ定義関数である。以下、想定したデータモデルの簡単な説明に続いて、モナドの利用とDB操作体系の実現について順に説明をし、最後にまとめを行う。

2　データモデル

型付けされた変数-値関連を抽象化したものをデータモデルとして用いる。型σの値全体をV_σ、σに関連付けられた識別子集合をI_σで表し、DBに格納された型の集合をΣとした時、このDBの状態集合は次のように定義される：

$$\{(s_\sigma)_{\sigma\in\Sigma} \mid s_\sigma \in I_\sigma \times V_\sigma\}$$

Implementing database operations for the Haskell programming language using Monad and class mechanism.
A. Kitamura, Y. Takeshima, Y. Ichikawa, I. Fujishiro and H. Sato,
Faculty of Science, Ochanomizu University.

```
data Student = Student String Int
        deriving (Text, Persistent)
mainIO
 = returnIO "university"      `startDB`
   allDB `thenDB` \vars ->
   findStudent vars "Kawada" `thenDB` \v ->
   updateDB v (Student "Kawada" 22)
                                `commitDB_`
   returnIO ()

findStudent vars name = .....
```

図．簡単な検索例

ここでs_σはσ型データの有限集合であり、$(s_\sigma)_{\sigma\in\Sigma}$において、識別子は一意でなくてはならない。また、$\sigma$はポリモルフィックであってはならない。基本的なDB操作としては、型$\sigma \in \Sigma$毎に、all_σ、new_σ / $delete_\sigma$、$update_\sigma$ / $refer_\sigma$を有し、各々、s_σ中の全ての識別子を検索する操作、(識別子, 値)対の挿入と削除を行う操作、変数への代入と参照に相当する操作である。ここで、new_σ操作では一意な識別子は自動的に割り当てられる。

3　順次実行制御のためのモナドの利用

DBモナドは(DB, returnDB, thenDB)なる代数系であり、単位半群(台集合, 単位元、合成演算子)に対応させて考えると見やすい。前節のDB状態をHaskell言語内部で表現した型をDBWorldとするとDB操作を表す多様型DBは次のように定義される：

```
type DB a = DBWorld -> (a, DBWorld)
```

ここで、aは型変数である。DB a型の関数は、DB状態を受けとってa型の操作結果と新しいDB状態を対にして返す。例えばall_σや$refer_\sigma$は、検索結果とDB状態の対を返す。

thenDBはDB操作を合成して新しいDB操作を作る。thenDB m kなるDB操作は、図式的には

$$w_0 \xrightarrow{m} (x_1, w_1)\,;\, w_1 \xrightarrow{k\ x_1} (x_2, w_2)$$

と表される。また、returnDBは任意の式からDB操作を作る関数で、returnDB eなるDB動作は、図

式的には次のように表される：

$$w \longrightarrow (e, w)$$

DB モナドの実装では、プログラム内および2次記憶上での DB 状態表現が問題となる。プログラム内では s_σ はリストを用いて表現し、また $(s_\sigma)_{\sigma\in\Sigma}$ も s_σ のリストを用いて表現した。この時、型システムの制約上、全ての s_σ は同一の型でなくてはならないため、(識別子, 値) の対ではなく (識別子, 値の文字列表現) の対を用いた。その結果、文字列表現可能なデータのみが永続性を持つことができ、関数を含む永続データは利用できない。2次記憶上では s_σ は1つの可読ファイルに記録され、ファイルと型の関係および識別子生成用情報はデータ辞書ファイルに記録される。これらのファイルは DB 毎にまとめられて1つのディレクトリに置かれる。

DB トランザクションは `startDB` によって開始され、`commitDB` または `rollbackDB` によって終了する。`startDB` は使用する DB に対応するディレクトリからファイルを読みとり、DB 状態、すなわち DB-World 型の値を構成する。`commitDB` は最終的な DB 状態をファイルに書き戻し、`rollbackDB` は DB 状態を破棄する。

4 DB 操作体系の実装

2節で示した DB 操作は多重定義関数として実現する。Haskell ではこれはクラス機構によって制御される。ここでクラスとは、いくつかの多重定義関数に関連付けられた型の集合であり、C をクラス、$f, g \cdots, h$ を関連付けられた多重定義関数、σ を C のインスタンスとすれば、プログラム内で次の関数が利用可能となる：

$$f_\sigma, g_\sigma, \cdots, h_\sigma$$

これらの関数をメソッドと呼ぶ。ここでは、先に述べた DB 操作 all_σ などに関連付けられたクラスとして、`Persistent` クラスを導入し、永続性を付加される型をこのクラスのインスタンスとする。

インスタンス定義には2つの方法があり、1つは、メソッドの定義をプログラム内でユーザが明記するものである。もう1つは、この定義を Haskell 処理系が自動導出するものであり、この場合ユーザはメソッドの詳細を考慮する必要がなくなる。

`Persistent` クラスの場合には、メソッド定義をユーザが行うことはプログラムの信頼性の低下を招く恐れがあり、また、このメソッド定義は種々の外部的な要因によって変更され得るものである。これらの観点から、`Persistent` クラスのメソッド定義については自動導出が望ましい。ところが、この自動導出機能は規定のクラスに対してしか使えないため、今回我々は処理系に手を加え、`Persistent` クラスに対しても自動導出を可能にした。

これにより、図 の `Student` のように型宣言を行うことで、必要な DB 操作用の関数が使えるようになる。ここで `deriving` 節がメソッド定義の自動導出を指示している。

5 まとめ

関数型算譜言語 Haskell 上に、モナドによる順次実行制御と `Persistent` クラスを用いて DB 操作体系の導入を行った。基本操作は固定であるが、これらを組み合わせて任意の操作が作成できるという意味で拡張性を有している。また、モジュール機構と組み合わせることで、目的に合わせて柔軟に DB インターフェースの提供が可能である。

問題点としては次のような点が挙がられる：(1) DB 状態が文字列ベースのため、関数を含む永続データが作成できず、型と永続性の直行性 [6] が崩れている；(2) DB 状態は全て主記憶上に読み込むため、大規模な DB は取り扱うことができない；(3) スキーマ変更のことを考慮していない；および、(4) 同時実行制御機構が無い。今後はこれらの問題点を改善すると同時に、データモデル面でのタイプシステムも含めた再検討、既存のデータベース管理システムとのインターフェースの実装、永続記憶の利用、応用プログラムの作成などを行う予定である。

参考文献

[1] Hamond, K., et al., "Improving Persistent Data Manipulation for Functional Language," in [7].

[2] Peyton Jones, S. L. and P. Wadler, "Imperative Functional Programming," *ACM POPL*, 1993.

[3] Nikhil, R. S., "The Semantics of Update in a Functional Programming Language," in *Advances in Database Programming Languages*, F. Bancilhon and P. Buneman eds., ACM Press, 1990.

[4] Hudak, P., S. L. Peyton Jones and P. Wadler, eds., "Report on the Functional Programming Language Haskell, Version 1.2," *ACM SIGPLAN Notices*, **27(5)**, May 1992.

[5] Hall, C. et al., "Glasgow Haskell Compiler: A Retrospective," in [7].

[6] Atkinson, M. and P. Buneman, "Types and Persistence in Database Programming Langaues," *ACM Comp. Surv.*, **19(2)**, Jun. 1987.

[7] Launchbury, J. and P. Sansom, eds., *Functional Programming, Glasgow 1992*, Springer-Verlag, 1993.