修士学位論文要約（平成29年 3 月）

ML 系多相型言語への自然結合の導入の研究

佐々木 智啓
指導教員：大堀 淳

A Study on Integrating Natural Join into a ML-style Programming Language

Tomohiro SASAKI
Supervisor: Atsushi OHORI

Natural join can be regarded as an operation to combine two partial descriptions. This operation is useful for processing large scale data. The purpose of this paper is to integrate the natural join operation into a ML-style programming language. To achieve this, we develop a type system to introduce this operation as a polymorphic function. We also develop an evaluation model for this operation. This is done by introducing values with static information. The proposed system is implemented by extending the SML# compiler.

1. はじめに

自然結合演算は関係代数上の演算であり，関係代数に基づく関係データベース上では 2 つのテーブルを共通するカラムで結合する演算とされている．この演算は Buneman らにより，上限を求める演算の特殊な場合であるという一般化された解釈が与えられた [2]．この演算は特に大規模なデータを扱う上で有益な演算であり，ML 系多相型言語において組み込みの演算として提供できるならば言語の可能性をより広げることを期待できる．データベースプログラミング言語としての自然結合演算を含む言語システムの研究は Ohori らによって既にされている [4]．ただし，この研究では多相 LET 式など，現実のプログラミングで用いる言語構造が含まれていないという未解決の課題がある．本研究では，Ohori らによるシステムをベースとし，Odersky らによる HM(X)[3] スタイルの多相 LET 式を統合したシステムを構築し，自然結合演算を含む実用 ML 系多相型言語の構築に必要となるシステムを構築する．

2. 対象言語の定義

本研究で構築するシステムの対象言語の式 e，単相型 τ と多相型 σ はそれぞれ以下のように定義した．

$$
\begin{array}{rcl}
e & ::= & c^b \mid x \mid \lambda x.e \mid e\ e \mid \{l = e, \ldots, l = e\} \\
 & \mid & [e, \ldots, e] \mid \mathit{join}(e, e) \mid \mathit{let}\ x\ =\ e\ \mathit{in}\ e \\
\tau & ::= & b \mid t \mid \tau \to \tau \mid \{l : \tau, \ldots, l : \tau\} \mid \tau\ \mathit{list} \\
\sigma & ::= & \forall \overline{t}.C \Rightarrow \tau
\end{array}
$$

ここで，b は定数型，c^b は型 b の定数，x は変数名，t は型変数，C は制約集合を表すメタ変数である．式 $\mathit{join}(e_1, e_2)$ は e_1 と e_2 の自然結合結果を求める式として導入している．式 $[e, \ldots, e]$ および対応する型 $\tau\ \mathit{list}$ は Ohori らによる研究における集合を模倣するものとして導入している．制約集合 C の要素 c は $\tau = \tau \sqcup \tau$ の形をとる．また，制約 c が $\tau_1 = \tau_2 \sqcup \tau_3$ を表しているとき，τ_1 は τ_2 と τ_3 の型上の上限であることを表す．本論文で構築するシステムにおいては，制約は型変数に対する代入の妥当性に関する述語として作用するものである．

3. 型システムの構築

型変数から多相型への写像である型環境 Γ と制約集合 C の下で式 e の型が τ であることを $C, \Gamma \vdash e : \tau$ と表し，型規則と呼ぶ．型規則は前提がある場合は水平線の上に前提を示し，下に帰結を示す．Ohori らによるシステム中の型規則には制約集合は現れなかったが，HM(X) を統合する上で今回型規則中に現れるようになった．本研究で構築したシステムの型規則のうち，制約集合が関連するものについて示す．

$$
\frac{[\overline{\tau}/\overline{t}]C' \subseteq C \qquad \tau' = [\overline{\tau}/\overline{t}]\tau \qquad \text{for some } \overline{\tau}}{C, \Gamma\{x : \forall \overline{t}.C' \Rightarrow \tau\} \vdash x : \tau'}
$$

$$
\frac{C, \Gamma \vdash e_1 : \tau_1 \qquad C, \Gamma \vdash e_2 : \tau_2 \qquad (\tau = \tau_1 \sqcup \tau_2) \in C}{C, \Gamma \vdash \mathit{join}(e_1, e_2) : \tau}
$$

$$
\frac{\begin{array}{c} C \cup C', \Gamma \vdash e_1 : \tau_1 \qquad C, \Gamma\{x : \forall \overline{t}.C' \Rightarrow \tau_1\} \vdash e_2 : \tau \\ (\mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(C)) \cap \overline{t} = \emptyset \qquad \mathrm{FTV}(C') \subseteq \overline{t} \end{array}}{C, \Gamma \vdash \mathit{let}\ x\ =\ e_1\ \mathit{in}\ e_2 : \tau}
$$

また，定義した型規則に対し，型推論アルゴリズム $\mathcal{WC}$ を定義した．$\mathcal{WC}$ は制約集合，型環境，式の 3 つを受け取り，制約集合，代入，単相型の 3 つを返すものである．定義した型推論アルゴリズムのうち，前述した 3 つの型規則に対応するものについて示す．

© 2017 東北大学電気通信研究所

$$
\begin{aligned}
&\mathcal{WC}(C,\Gamma,x) = \text{let } (\forall\overline{t}.C' \Rightarrow \tau) = \Gamma(x)\\
&\qquad\qquad S = [\overline{t'}/\overline{t}] \quad (\overline{t'}\ \textit{fresh})\\
&\qquad\qquad \text{in } (C \cup S(C'), \emptyset, S(\tau))\\
&\mathcal{WC}(C,\Gamma,\mathit{join}(e_1,e_2)) =\\
&\quad \text{let } (C_1,S_1,\tau_1) = \mathcal{WC}(C,\Gamma,e_1)\\
&\qquad (C_2,S_2,\tau_2) = \mathcal{WC}(C_1,S_1(\Gamma),e_2)\\
&\quad \text{in } (C_2 \cup \{t = S_2(\tau_1) \sqcup \tau_2\}, S_2 \circ S_1, t) \quad (t\ \textit{fresh})\\
&\mathcal{WC}(C,\Gamma,\mathit{let}\ x\ =\ e_1\ \mathit{in}\ e_2) =\\
&\quad \text{let } (C_1,S_1,\tau_1) = \mathcal{WC}(C,\Gamma,e_1)\\
&\qquad (C_1',\sigma) = \mathit{Cls}(C_1,S_1(\Gamma),\tau_1)\\
&\qquad (C_2,S_2,\tau_2) = \mathcal{WC}(C_1',S_1(\Gamma)\{x:\sigma\},e_2)\\
&\quad \text{in } (C_2, S_2 \circ S_1, \tau_2)
\end{aligned}
$$

定義した型推論アルゴリズムに対しては，以下の健全性定理が成り立つ．

定理 3.1 もし，$\mathcal{WC}(C,\Gamma,e) = (C',S,\tau)$ ならば，$S(C) \subseteq C'$ かつ，$C', S(\Gamma) \vdash e : \tau$ が成り立つ．

4. 値の評価モデルの構築

Buneman らによる一般化された自然結合演算の定義に基づいて自然結合を定義する場合，静的な情報，すなわち型の情報が必須である．ただし，この情報はコンパイル後の実行時コードにおいては不要な情報が多く，実用プログラミング言語においては静的な情報の多くはコンパイル時のみ使用し，実行時コードには残っていない場合が多い．このような背景から，本研究では Abadi らによる Dynamic[1] の概念を参考にし，静的な情報を含む値を定義し，静的な情報を含む値と含まない値の間での相互変換の枠組みを形式的に定義した．自然結合式の値の評価モデルにおいては，一度静的な情報を含む値へと変換してから上限を計算，再度静的な情報を含まない値へと変換するということを行っている．

5. 実装

自然結合演算の実装は ML 系多相型言語のひとつである SML# を拡張することで行った．以下は実装後の SML# コンパイラの対話型モードにおける自然結合演算の実行例である．

```
# fun f (x, y) = _join (x, y);
val f = fn
  : ['a, 'b, 'c. ('c = 'a join 'b) =>
     'a * 'b -> 'c]
# f ({a = 1, b = "hoge"},
>    {b = "hoge", c = 1.1});
val it = {a = 1, b = "hoge", c = 1.1}
  : {a: int, b: string, c: real}
# f ({a = 1,
>     b = {ba = true, bb = "fuga"}},
>    {b = {bb = "fuga",
>          bc = fn x => x + 1},
>     c = 1.1});
val it =
  {a = 1,
   b = {ba = true,
        bb = "fuga",
        bc = fn},
   c = 1.1}
  : {a: int,
     b: {ba: bool,
         bb: string,
         bc: int -> int},
     c: real}
```

6. まとめ

本研究では，ML 系多相型言語に自然結合演算を導入するために必要となる型システムおよび値の評価モデルについて構築した．型システムについては Ohori らによるシステムに Odersky らによる HM(X) スタイルの多相 LET 式を統合することで構築した．値の評価モデルについては，Abadi らによる Dynamic の概念を参考とした静的な情報を含む値を導入することで，実行時に必要となる情報を得られるようなモデルを構築した．さらに，ML 系多相型言語のひとつである SML# コンパイラを拡張することで構築したものを実装し，実用 ML 系多相型言語で構築したものが実現可能であることを確認した．

参考文献

1) Martin Abadi, Luca Cardelli, Benjamin Pierce, and Gordon Plotkin. Dynamic typing in a statically-typed language. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 213–227, New York, NY, USA, 1989. ACM.

2) Peter Buneman, Achim Jung, and Atsushi Ohori. Using powerdomains to generalize relational databases. *Theor. Comput. Sci.*, 91(1):23–55, December 1991.

3) Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. *Theor. Pract. Object Syst.*, 5(1):35–55, January 1999.

4) Atsushi Ohori and Peter Buneman. Type inference in a database programming language. In *Proceedings of the 1988 ACM Conference on LISP and Functional Programming*, LFP '88, pages 174–183, New York, NY, USA, 1988. ACM.