# 自然結合制約を含む型推論アルゴリズムの実装方式

佐々木 智啓　上野 雄大　大堀 淳

本稿では，型付き関数型言語のコンパイラに自然結合演算を加える実装方式について報告する．自然結合は強力なレコード演算のひとつであり，関係データベースにおける基本演算のひとつでもある．レコードを含む型付き言語が自然結合をサポートするならば，より柔軟かつ記述力の高いレコード処理が書けるようになるとともに，SQL とのより高い親和性も実現できると期待できる．この目的のため，著者らは SML#に自然結合演算を加え，その型推論器を実装した．本稿では，まず自然結合を含む型システムの概要を紹介したのち，行った実装の詳細と利用例を報告し，今後の展望について述べる．

## 1　はじめに

自然結合は関係代数において部分的な 2 つの記述を統合する演算と解釈できる．関係データベースにおいては 2 つのテーブルを結合する基本演算のひとつとして提供されている．以下は関係データベース上での自然結合演算 $\bowtie$ の例である．

| 名前 | id |
|---|---|
| 太郎 | 1 |
| 花子 | 2 |
| 一郎 | 3 |

$\bowtie$

| id | 給料 |
|---|---|
| 1 | 450 |
| 2 | 500 |
| 4 | 700 |

$=$

| 名前 | id | 給料 |
|---|---|---|
| 太郎 | 1 | 450 |
| 花子 | 2 | 500 |

文献 [1] によれば，自然結合演算は 2 つの値の上限を計算する演算の特殊な場合であると解釈できる．したがって，適切な順序関係を定められるならば，関係データベース以外の処理系でも自然結合演算を提供できるはずである．

この一般化された自然結合演算を ML 系関数型言語に組み込みの演算として提供できるならば，複雑なデータを取り扱うプログラムをより効率よく記述できるようになり，ML 系関数型言語の可能性を大きく広げると期待できる．また，提供された演算と SQL の ML への統合方式 [5] を組み合わせれば，SQL の自然結合を ML に型付きで統合することも可能である．

自然結合演算の ML 系関数型言語への組み込みに向け，著者らは自然結合演算を含む ML の型システムの開発に取り組んできた [7]．この型システムは文献 [3] の型システムをベースとして作られている．また，LET 多相などの文献 [3] の型システムで扱っていない箇所を補うために，HM(X) 型システム [2] の制約を型システムに加える考え方を加えて導入している．文献 [7] で示した型システムは従来の ML 系型システムとほぼ直行しており，ML 系型システムを採用する実際のコンパイラに容易に実装できると思われる．

著者らは文献 [7] の型システムを SML# コンパイラ 3.1.0 版 [8] を拡張することで実装した．実装の過

---

An implementation method of type inference algorithm including natural join constraints

Tomohiro Sasaki, 東北大学大学院情報科学研究科, Graduate School of Information Sciences, Tohoku University.

Katsuhiro Ueno, Atsushi Ohori, 東北大学電気通信研究所, Research Institute of Electrical Communication Tohoku University.

程では文献[7]で述べられていない種々の課題が発生した．その主要な要因はランク 1 多相性である．ランク 1 多相型をうまく推論するために，SML#コンパイラの型推論アルゴリズムは様々な構文の型推論時に多相型に対する多様な処理を行なっている．今回実装する型システムでは，多相型は束縛する型変数に閉じた制約も束縛している．したがって，変数式，LET 式，JOIN 式のみで制約を取り扱うのではなく，多くの箇所で制約を取り扱う必要があった．本稿では文献[7]で提案された型推論器の実装と技術的な課題の詳細を報告する．

自然結合を SML#に導入する具体的な内容は，以下の構文を加えることである．

$$\_\texttt{join}(e_1, e_2)$$

この構文はレコード $e_1$ と $e_2$ を自然結合したレコードを表す．構文を完成させるためには型推論器とコード生成器の拡張が必須である．拡張した SML#コンパイラはこの構文に文献[7]で与えられた通りの多相型を与える．文献[7]では型システムについては言及している一方，多相的な自然結合の評価モデルの構築は将来の課題としている．そのため現段階では結果のレコードを計算するコード生成は未実装であり，この構文を実行すると例外が発生するようにしている．SQL を ML に統合する際に必要となる toy term の生成にはこれで十分であり，したがって自然結合を含む SQL クエリの ML への統合は可能である．以上の機能を実現するため，SML#コンパイラの型推論器までのフェーズを拡張した．

行った拡張はほぼ型推論器で閉じている．本稿では SML#の型推論器に制約に基づく型規則を導入するための方針とその実装の詳細を述べる．さらに，本稿では，より完全な自然結合演算の ML への統合に向けた展望を述べる．

本稿の構成は以下の通りである．2 節では文献[7]の型システムの概略を述べる．3 節では SML#におけるランク 1 多相性の概略を述べる．4 節では型システムの SML#コンパイラへの実装の詳細を述べる．5 節で結論と今後の展望，特により完全な自然結合演算の ML への統合に向けた課題を述べる．

## 2 自然結合演算の型付け

本節では，文献[7]が提案する自然結合演算に関する制約ベースの型システムの概要を述べる．

型付け規則を自然結合制約で拡張するために，自然結合制約の導入，型ジャッジメントの拡張，多相型の拡張が行われた．自然結合制約はオペランドと結果の関係を示し，その形は $\tau = \tau \sqcup \tau$ の形である．型ジャッジメントは型環境 $\Gamma$ と式 $e$ から型 $\tau$ を導出する形を制約集合 $C$ で拡張し，$C, \Gamma \vdash e : \tau$ の形で示すようにした．多相型 $\sigma$ は束縛した変数に閉じている制約集合を加え，$\forall \bar{t}.C \Rightarrow \tau$ の形で示すようにした．

式に対する型付け規則のうち，制約が関連するものは以下のように示された．

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

$$\frac{C_1 \cup C_2, \Gamma \vdash e_1 : \tau_1 \quad C_1, \Gamma\{x : \forall \bar{t}.C_2 \Rightarrow \tau\} \vdash e_2 : \tau \quad (FTV(\Gamma) \cup FTV(C_1)) \cap \bar{t} = \emptyset, FTV(C_2) \subseteq \bar{t}}{C_1, \Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau}$$

変数 $x$ に対する型付け規則は $x$ に対応する多相型に閉じている制約集合をインスタンス化し，インスタンス化された制約集合が型ジャッジメントにおける制約集合の部分集合であるという条件を加えている．

文献中に示された自然結合制約で型推論アルゴリズムを拡張するためのアイデアを以下に述べる．まず，制約集合はグローバルなものとみなす．そのため，LET 式のケースで閉じている制約を多相型に束縛させる場合を除き，制約集合は常に和をとる．また，制約が増えるのは JOIN 式または変数式のみである．得られる制約集合は LET 式の型推論時に解き，トップレベルが LET 式以外の式は等価な LET 式とみなすことでトップレベルで制約を解くようにする．また，制約集合を解く場合も計算時間の増加を抑えるためにオペランドの型に自由変数が存在しない制約のみを解く．LET 式のケースで多相型に束縛しない変数は型環境 $\Gamma$ から制約集合 $C$ をたどって到達できる変数とする．

## 3 ランク 1 多相性

SML#コンパイラが採用するランク 1 多相性を持つ型システムは文献 [6] に示されている型システムを基としている．また，SML#におけるランク 1 多相の概略は SML#のマニュアルにも記述されている．本節では，4 節を読む上で前提となる，SML#コンパイラが採用する型システムの概略を述べる．

SML#コンパイラで扱う単相型 $\tau$，多相型 $\sigma$ はそれぞれ以下のように定義される．

$$\tau \quad ::= \quad t \mid b \mid \tau \to \tau \mid \cdots$$
$$\sigma \quad ::= \quad \tau \mid \forall \bar{t}.\sigma \mid \tau \to \sigma \mid \cdots$$

多相型が部分型に現れるため，多相型の生成が LET 式のみではなく関数抽象式でも行われる．

ランク 1 多相性を導入することで従来の型システムから大きく変わる関数適用式について述べる．この型システムにおける関数適用式の型付け規則を以下に示す．

$$\frac{\Gamma \vdash e_1 : \tau \to \sigma \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1\ e_2 : \sigma}$$

これを踏まえた上で関数適用式 $e_1\ e_2$ に対する型推論アルゴリズム $\mathcal{W}$ で起こることを述べる．まず，式 $e_1$ に対し $\mathcal{W}$ を適用し，$e_1$ の型 $\sigma_1$ を得る．型付け規則から，なんらかの $\tau_1$，$\sigma_2$ に対し，$\sigma_1$ と $\tau_1 \to \sigma_2$ を単一化する必要がある．従来の単一化アルゴリズム $\mathcal{U}$ は多相型に対して適用できず，単一化対象の型にも多相型が含まれているため，$\mathcal{U}$ を用いる以外の手段で単一化する必要がある．提案された型推論アルゴリズムでは，$\sigma_1$ の形に応じて型の一部分をインスタンス化することで単一化としている．また，式 $e_2$ に対し $\mathcal{W}$ を適用し，得られた $e_2$ の型 $\sigma_3$ の fresh な型変数による単相型のインスタンス $\tau_2$ を生成し，$\tau_1$ と $\tau_2$ の単一化を行なっている．

## 4 型推論アルゴリズムの実装

著者らは文献 [7] で提案された自然結合制約を含む型推論アルゴリズムを SML#コンパイラの型推論器を拡張することで実装した．3 節で述べたランク 1 多相性により，提案されたアルゴリズムでそのまま型推論器を拡張することはできない．本節では，SML#コンパイラの型推論器におけるランク 1 多相性の扱いに触れた上で，型推論器の主な変更点と，実装後の型推論器を用いた型推論の例を述べる．

SML#コンパイラの型推論器は，ランク 1 多相性に関してはおおむね文献 [6] で示されたアルゴリズムをそのまま実装している．型推論器内では多相型に対する処理は大まかに generalization，instantiation，coercion に分けられ，型推論器中の様々な箇所に発生する．ここで，coercion は多相型を部分に含む型を等価な型にする処理の SML#コンパイラ内での呼び方である．これらの説明と変更点を以下で述べる．

generalization は与えられた型に閉じている型変数を束縛し，多相型を作る処理である．演算を行わない関数式等の式，LET 式の型推論時に多相型を生成する際に行われる．閉じている型変数の導出は与えられた型から到達可能な型変数から，型環境から到達可能な型変数を除外することで行う．

generalization に対しては，以下の 2 つの変更を行った．まず，多相型で束縛するものはその中に閉じているという性質の保持のために，導出時に現在の制約集合もたどるよう変更した．また，多相型では束縛した変数に閉じている制約も束縛しているため，現在の制約集合中から束縛対象の変数に閉じている制約も導出す 1 るよう変更した．

instantiation は多相型が束縛している型変数に新しい型変数を割り当て，単相型とする処理である．変数の型推論時や $\mathcal{U}$ を使用するために多相型のインスタンスを生成する際に行われる．新しい型変数の割り当ては，束縛している型変数から新しい型変数への代入を作成し，その代入を適用することで行う．

instantiation に対しては，インスタンス化対象の多相型が束縛している制約を制約集合に追加する変更を行った．多相型で束縛している制約も満たすべき制約であるため，束縛している制約にも作成した代入を適用し，制約集合に追加するよう変更した．

coercion はある型を型推論を進める上で必要な型に一致させる処理である．3 節で述べた関数適用式の型推論時のように，部分に含まれる多相型を等価な単相型に変換する処理も行われる．

coercion に対しては，変換される型が多相型の場

合，その多相型に含まれる制約も満たすべき制約であるので，この制約も制約集合に追加するよう変更した．

また，提案されたアルゴリズムでは LET 式及びトップレベルで制約を解くようにしていた．アルゴリズムの方針に従った上で多相型が制約を束縛する前に制約を解かせるため，generalization 前とトップレベルに制約を解く処理を追加した．

最後に，実装後の型推論の実例を示す．まず，SQL の自然結合に型付けした例を示す．SML#は SQL をプログラムへ統合する機能を有しており，この機能に自然結合を追加した．対話型モードで SQL の自然結合の型を推論したものを以下に示す．

```
# _sql db =>
>   select #t.name, #t.age
>   from (#db.t1 natural join #db.t2) as t;
val it = fn
  : ['a#{t1: 'b, t2: 'c}, 'b#{}, 'c#{},
     'd, 'e, 'f, 'g,
     'h#{age: 'f, name: 'd}.
       ('h = 'b join 'c) =>
       'a db -> ('d * 'f) query]
```

推論結果の型は，データベース db は少なくとも 2 つのテーブル t1 と t2 を有しており，t1 と t2 の自然結合の結果は少なくとも 2 つの列 age と name を有することを示している．次に，制約を満たす型が存在しないが，型推論が行えてしまう例を以下に示す．

```
# fn (x, y) =>
>   (#a x ^ "a", #a y + 1, _join (x, y));
val it = fn
  : ['a#{a: string}, 'b#{a: int}, 'c#{}.
       ('c = 'a join 'b) =>
       'a * 'b -> string * int * 'c]
```

このとき，制約を満たす x と y の型は存在しないが，制約の妥当性の確認はオペランドの自由変数が存在しなくなるまで行わないため型推論は行えてしまう．

## 5 まとめと今後の展望

本稿では，文献[7] で報告している自然結合制約を含む型推論アルゴリズムの実装を報告した．既存の型推論アルゴリズムに対する自然結合制約による拡張はそのアルゴリズムの性質によっては文献で言及されている以外の課題が存在するが，拡張を適切に行えば実現できる．また，型推論器の拡張による変更の範囲はほぼ型推論器の実装に閉じていることが確認できた．実際に SML#コンパイラの型推論器を自然結合制約で拡張した例を報告し，利用可能であることも確認できた．

今後の課題は自然結合の計算を実現することである．一般化された自然結合の意味は文献[1] で示されている．自然結合を言語に導入する観点では文献[4] で導入の概略が示されている．一方で，これらを現実のコンパイラへ導入するには型主導コンパイルにおける扱いといった種々の課題が存在する．

**謝辞** 本研究の一部は JSPS 科研費 25280019「ML 系多相型言語 SML#の実用化技術に関する基礎研究」および 15K15964「実用プログラミング言語のための系統的言語開発基盤の実現」の助成を受けて実施されたものである．

## 参 考 文 献

[1] Buneman, P., Jung, A., and Ohori, A.: Using Powerdomains to Generalize Relational Databases, *Theoretical Computer Science*, Vol. 91, No. 1(1991), pp. 23–56.

[2] Odersky, M., Sulzmann, M., and Wehr, M.: Type Inference with Constrained Types, *Theor. Pract. Object Syst.*, Vol. 5, No. 1(1999), pp. 35–55.

[3] Ohori, A. and Buneman, P.: Type Inference in a Database Programming Language, *Proceedings of the 1988 ACM Conference on LISP and Functional Programming*, LFP '88, New York, NY, USA, ACM, 1988, pp. 174–183.

[4] Ohori, A., Buneman, P., and Breazu-Tannen, V.: Database Programming in Machiavelli&Mdash;a Polymorphic Language with Static Type Inference, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, SIGMOD '89, New York, NY, USA, ACM, 1989, pp. 46–57.

[5] Ohori, A. and Ueno, K.: Making Standard ML a Practical Database Programming Language, *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, New York, NY, USA, ACM, 2011, pp. 307–319.

[6] Ohori, A. and Yoshida, N.: Type Inference with Rank 1 Polymorphism for Type-directed Compilation of ML, *SIGPLAN Not.*, Vol. 34, No. 9(1999),

pp. 160–171.
[7] Sasaki, T., Ueno, K., and Ohori, A.: SML# with Natural Join, *ACM SIGPLAN Workshop on ML*, 2016. to appear.
[8] SML# Compiler. http://www.pllab.riec.tohoku.ac.jp/smlsharp/