Agda による定式化された型推論器の実装と拡張

門脇香子　(指導教員：浅井健一)

1　本研究の貢献

McBride は、型変数の数を巧妙に管理することで、停止性が明らかな形で型の unification を表現できることを示した [1]。そこで我々は、この理論をもとにして unification を実装し、正当性の証明のついた型推論器 infer を実装した [3]。

しかし、この型推論は主に 2 つの問題があった。1 つ目は、型変数の数を count 関数で数え、型変数の数が矛盾しない証明を巧妙に管理する必要があることである。つまり、この型推論では構文に応じて型変数の数を人が厳密に数えて証明を実装する必要があり、実装が困難になるという問題があった。2 つ目の問題は、これまでの型定義の手法では、構文を拡張しようとする際、別の型推論を実装し、その正当性を示さねばならなかった。この主な 2 つの問題を回避するため、今回の型推論器では、型変数を数ではなく不等式を持ち歩くことで管理する手法を採用した。また、generic programming [2] の考え方を取り入れ、型の定義をユーザが与えると、指定した型に対してそのまま型推論をできるような手法を採用した。これらの手法により、型変数の数を厳密に管理しなくても、Agda に指定された不等式を解けば良くなり、既存研究では非常に困難であった他の型も容易に導入できるようになった。

本研究の貢献は、McBride が示した unification の機構 [1] をもとにして、正当性の証明がついた型推論器を実装し、不等式による型変数の管理や generic programming の手法を用いて型変数の数の管理を容易にするとともに、構文や型の拡張をより簡単にしたことである。

2　型と構文の定義

型定義は以下のように表現する。

$$\tau \quad = \quad m \mid \tau_1 \to \tau_2$$

項は以下のように表現する。

$$M \quad = \quad x \mid \lambda x.\, M_1 \mid M_1\, M_2$$

型判断は以下の通りに行う。

$$\Gamma \vdash M : \tau$$

また、型規則は以下のようになる。

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{\Gamma, x : \tau_2 \vdash M_1 : \tau_1}{\Gamma \vdash \lambda x.\, M_1 : \tau_2 \to \tau_1}$$

$$\frac{\Gamma \vdash M_1 : \tau_2 \to \tau_1 \quad \Gamma \vdash M_2 : \tau_2}{\Gamma \vdash M_1\, M_2 : \tau_1}$$

2.1　構文定義

変数の出現位置に関して de Bruijn index を用いている。

入力される構文として、自由変数の数が n 個の well-scoped な項として以下を定義する。

```
- 自由変数の数が n 個の well-scope な項
data WellScopedTerm (n : ℕ) : Set where
  Var : Fin n → WellScopedTerm n
  Lam : (s : WellScopedTerm (suc n)) → WellScopedTerm n
  App : (s1 : WellScopedTerm n) →
        (s2 : WellScopedTerm n) → WellScopedTerm n
```

出力される構文として、自由変数の数が n 個、型が t であるような well-typed な項を定義する。

```
- WellTypedTerm Γ t :  自由変数の数が n 個
- 型が t であるような well-typed な項
data WellTypedTerm {m n : ℕ} (Γ : Cxt n) :
       Type m → Set where
  Var : (x : Fin n) → WellTypedTerm Γ (lookup x Γ)
  Lam : (t : Type m) → {t' : Type m}
     → WellTypedTerm (t :: Γ) t'
       → WellTypedTerm Γ (t ⇒ t')
  App : {t t' : Type m} → WellTypedTerm Γ (t ⇒ t')
     → WellTypedTerm Γ t
       → WellTypedTerm Γ t'
```

本研究の型推論では、well-scoped な term を引数に取り型と必要な代入、対応する well-typed な term を返す関数 infer を作ることを目的とする。

3　不等式を用いた代入

本研究では、代入の型として AListType を用いる。AListType m'' m' は m'' 個の型変数を持つ型を m' 個の型変数を持つ型にする代入であり、ある m'' と m' が存在し、m' が m'' より大きくないという条件さえ満たされていれば代入が可能である。この条件を持ち歩くことにより、型変数を事細かに計算する必要がなくなった。

4　Unification の定式化

本研究では、型推論において必要な unification の手法として McBride の示した手法を Agda で実装したものを用いる。また、McBride の手法に加え、well-typed な型推論器の実装においては、unification の際に単一化を行う代入 σ を得るのみでなく、「確かに単一化されている」という証明も返すことが必要となった。

本研究では、必要な代入とともにこの証明を返す mgu2 を実装した。

5　型推論

5.1　infer 関数の定義

infer 関数の型定義は以下のようになる。

```
infer : (m : ℕ) → {n : ℕ} → (Γ : Cxt {m} n) →
   (s : WellScopedTerm n) →
Maybe (Σ [ m'' ∈ ℕ ]
   Σ [ m' ∈ ℕ ]
   Σ [ m≤m'' ∈ m ≤ m'' ]
   Σ [ σ ∈ AListType m'' m' ]
   Σ [ τ ∈ Type m' ]
   Σ [ w ∈ WellTypedTerm (substCxt≤ σ m≤m'' Γ) τ ]
   erase w ≡ s)
```

この infer 関数は以下のものを入出力する。

入力

- s : m 個の型変数をもつ WellScopedTerm
- Γ : 型環境

出力

- m'' , $m' : Nat$
- $m \leq m'' : m \leq m''$ であることの証明
- $\sigma : m''$ 個の型変数を持つ型を m' 個の型変数を持つ型にする代入
- τ : 代入後の型 (Type m')
- t : WellTypedTerm Γ' τ
- $erases \equiv w$: 返された well-typed term s から型情報を削除したものが、もとの well-scoped term と等しいという証明

ここで Γ' は Γ を不等式 $m \leq m''$ により m'' まで持ち上げた後、代入 σ を行い、型変数の数を m' に落としたものである。

5.2 Var

Var x の場合は、単に型環境の型を返せばよいので、返す型は $\phi[\downarrow_m^m](\Gamma^{(m)}(x))$ 、返す代入は $\phi[\downarrow_m^m]$ (空の代入)、返す term は Var x となる。ただし、「型環境に対して空の代入を適用したものがもとの型環境と同一である」という証明が必要になる。

5.3 Lam

ラムダ式の場合は、まず新しい型変数 $m^{(m+1)}$ を型環境 Γ に加えたうえで M の型推論を行い、s に型が付いた場合のみ、「$m^{(m+1)} \to$ M の型」が全体の型となる。

「型環境中の型変数の数を一気に持ち上げても、2 回に分けて持ち上げても同じ」という証明が必要になるが、これはまず型環境の中のそれぞれの型 $\tau^{(m)}$ に対してこの性質を証明し、その証明を型環境に対して再帰的に与えてやればよい。

5.4 App

App M_1 M_2 の型推論では、まず M_1 を型推論し、得られた代入 σ_1 をかけた型環境下で M_2 を型推論する。

型推論が成功した場合のみ、$\sigma_1\tau_1$ は $\tau_2 \to \beta$ の形をしているはずなので、unification を行うと、成功した場合のみ代入 σ_3 と確かに unification されることの証明 eq が得られる。

これを用いると、App の型規則から必要な型を得ることができ、ここで「複数の代入を順に施すのは、代入を結合したものを 1 度に施すのと同じである」ことの証明が必要になる。これは、有限集合の型である Fin 型の値を lift する inject 関数に対して幾つかの性質を示すことで、示すことが出来る。ただし、ここで関数の外延性を仮定することに注意したい。

6 構文の拡張

Fst, Snd, Cons の構文を拡張するにあたって、以下の手法を採用した。

6.1 より一般的な型への拡張

単純型付き λ 計算に対する型推論のみが必要なのであれば、関数の型のみ扱えるような型推論を作れば良いが、いろいろな体系に対する型推論を作ろうと思えば多様な型を扱える必要がある。しかし、それぞれの体系ごとに別の型推論を実装し、その正当性を示すのは現実的ではない。そこで、ここでは generic programming [2] の考え方を取り入れて、`fold` 関数等を用いて、ユーザが指定する任意の型を扱えるように全体を構築した。

ただし、ここで用いる `fold` 関数はデータ構造の有限性から停止性が明らかであるが、Agda の停止性チェッカでは停止性を確認できない。そのため、`fold` 関数の定義では停止性チェックを行わないように指示している。

6.2 追加した型規則

Fst, Snd, Cons を加えるにあたって追加した型規則は以下のようになる。

$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \Gamma \vdash M_2 : \tau_2}{\Gamma \vdash Cons\ M_1\ M_2 : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash Fst\ M : \tau_1} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash Snd\ M : \tau_2}$$

generic programming の考え方を用いることで、unification 部分を拡張することなく Fst, Snd, Cons を対象の構文に追加することができた。

7 今後の課題

今後の課題として、1 つ目はより syntax を拡張すること（例えば、 多相の Let 文など）である。2 つ目は、完全性の証明を加えることである。例えば現時点の infer 関数の型定義では常に推論に失敗するような infer 関数を実装することもできてしまう。つまり、型推論が失敗した時に、入力された構文に対応する型付けされた構文が存在しないという証明、式にすると

¬(∃ w : WellTypedTerm (erase $w \equiv s$))

を返す infer を実装することが必要になる。

参考文献

[1] McBride, C. “First-order unification by structural recursion,” *Journal of Functional Programming*, Vol. 13, No. 6, pp. 1061–1075, Cambridge University Press (November 2003).

[2] van Noort, T., A. R. Yakushev, S. Holdermans, J. Jeuring, and B. Heeren “A lightweight approach to datatype-generic rewriting,” *Proceedings of the ACM SIGPLAN Workshop on Generic Programming (WGP'08)*, pp. 13–24 (September 2008).

[3] 門脇 香子, 浅井 健一「Agda による型推論器の定式化」第１７回プログラミングおよびプログラミング言語ワークショップ論文集, 13 ページ (2015 年 3 月).