依存型意味論の証明探索とその実装

理学専攻・情報科学コース　佐藤 未歩

1　はじめに：依存型意味論と照応・前提

依存型意味論 (Dependent Type Semantics, 以下 DTS)[1] は、依存型理論 (Dependent Type Theory)[4] に基づく自然言語の証明論的意味論である。文の表す命題に依存型理論の型を対応づけており、Π 型 $(x{:}A) \to B$ と Σ 型 $\begin{bmatrix} x{:}A \\ B \end{bmatrix}$ を用いることによって文脈に依存した文の意味を表示することができる。

DTS では、照応・前提現象を未指定項 (underspecified term) によって記述する。たとえば、*A man entered. He whistled.* という 2 文を組み合わせた意味表示は、以下の図 1 のようになる。図 1 中の @ が未指定項であり、

$$\begin{bmatrix} v{:}\begin{bmatrix} u{:}\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix} \\ \mathbf{enter}(\pi_1(u)) \end{bmatrix} \\ \mathbf{whistle}\left(\pi_1(@ : \begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix})\right) \end{bmatrix}$$

図 1: *A man enterd. He whistled.* の意味表示

照応代名詞や前提トリガーなど、照応現象を引き起こす語彙項目の意味表示に含まれている。$@ : \begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix}$ の $\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix}$ は、@ に対するアノテーションである。

DTS を用いて照応解決・前提束縛の計算を行う際には、大きく分けて以下の 3 つの操作が必要である。

1. 文の意味表示に対する型チェック
2. @ の型に対する証明探索
3. 証明項による @ の置き換え

1 の型チェックでは、文の意味表示に対して型チェックを行うことで、$@ : \begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix}$ となるときの文脈を特定する。たとえば図 1 の例ならば、型チェックを行うことで $v : \begin{bmatrix} u{:}\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix} \\ \mathbf{enter}(\pi_1(u)) \end{bmatrix}$ という文脈のもとで、$@ : \begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix}$ であることが分かる。

2 の証明探索では、@ の型に対して証明探索を行うことで @ の型に対する証明項を見つける。図 1 の例ならば、@ の型である $\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix}$ という型を持つ項として $\pi_1(v)$ が存在し、証明探索を行うことでこの項を見つけることができる。このとき、1 において特定した $v : \begin{bmatrix} u{:}\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix} \\ \mathbf{enter}(\pi_1(u)) \end{bmatrix}$ という文脈を参照することができる。

3 は、2 で判明した証明項で文の意味表示中の @ を置き換える操作である。この操作を @-Elimination という。図 1 の例ならば、@ に対する証明項である $\pi_1(v)$ で @ を置き換えることによって、図 2 の意味表示を得ることができる。

$$\begin{bmatrix} v{:}\begin{bmatrix} u{:}\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix} \\ \mathbf{enter}(\pi_1(u)) \end{bmatrix} \\ \mathbf{whistle}\,(\pi_1(\pi_1(v))) \end{bmatrix}$$

図 2: @ 解消後の意味表示

このように、図 1 の意味表示から図 2 の意味表示を得ることで照応解決となる。

本研究では、DTS のための型チェック・証明探索アルゴリズムについて考察し、プログラミング言語 Haskell を用いて実装を行った。さらに、日本語 CCG パーザ lightblue[2] との接続を行うことで、日本語の文の入力に対して照応解決・前提束縛の計算を自動で行うことのできるシステムの構築を試みた。

2　型チェックと証明探索

2.1　DTS の型推論・型チェック

依存型理論の型推論は一般に undecidable であるため、依存型理論の型推論を行うには決定可能な部分体系を用いる方法が知られている。Löh による Agda[3] の型推論の体系では、アノテーション構文を用いて型推論の及ばない部分式の型をあらかじめ指定することで、依存型の部分体系に対する型チェックを decidable に行うことを可能にしている。

DTS のための型チェックアルゴリズムは佐藤・戸次 [6] によって定式化され、佐藤・戸次 [7] において実装されている。[6] では、DTS の体系を @ を含む体系と含まない体系の 2 つに分けて定義している。@ を含む体系を UDTT(Underspecified Dependent Type Theory)、含まない体系を DTT(Dependent Type Theory) という。文の意味表示に対する型チェックは UDTT のもとで行われる。UDTT の型規則（抜粋）を図 3 に示す。

UDTT の型規則の例として、(ΠI) 規則を以下の図 4 に示す。(ΠI) 規則は Π 型に対する導入規則であり、

$$\frac{\overset{\mathcal{D}}{\Gamma \vdash A :_{\downarrow} \mathsf{s}} \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma, x : A' \vdash M :_{\downarrow} B}{\Gamma \vdash \lambda x.M :_{\downarrow} (x{:}A) \to B}\ (\underline{\Pi I})$$

図 4: (ΠI) 規則

規則の上段が満たされたとき、項 $\lambda x.M$ は型 $(x{:}A) \to B$ を持つ、という意味である。規則中に現れる Γ は型環境を表す。規則の上段に現れている $[\![\mathcal{D}]\!]^{\mathsf{tm}}$ は @-Elimination を行っている箇所である。(ΠI) 規則では、型 $(x{:}A) \to B$ の A が型 type または kind を持つかど

$$\frac{\Gamma \vdash M :_\uparrow A}{\Gamma \vdash M :_\downarrow A}\,(\underline{CHK}) \qquad \frac{\overset{\mathcal{D}}{\Gamma \vdash A :_\downarrow \mathsf{s} \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma \vdash A' \ \mathit{true}}}{\Gamma \vdash (@ : A) :_\uparrow A}\,(\underline{@}) \qquad \frac{\overset{\mathcal{D}}{\Gamma \vdash A :_\downarrow \mathsf{s} \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma, x : A' \vdash M :_\downarrow B}}{\Gamma \vdash \lambda x.M :_\downarrow (x{:}A) \to B}\,(\underline{\Pi I}) \qquad \frac{\overset{\mathcal{D}}{\Gamma \vdash A :_\downarrow \mathsf{s}_1 \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma, x : A' \vdash B :_\downarrow \mathsf{s}_2}}{\Gamma \vdash (x{:}A) \to B :_\uparrow \mathsf{s}_2}\,(\underline{\Pi F})$$

$$\frac{\overset{\mathcal{D}}{\Gamma \vdash A :_\downarrow \mathsf{s}_1 \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma, x : A' \vdash B :_\downarrow \mathsf{s}_2}}{\Gamma \vdash \begin{bmatrix} x{:}A \\ B \end{bmatrix} :_\uparrow \mathsf{s}_2}\,(\underline{\Sigma F}) \qquad \frac{\overset{\mathcal{D}}{\Gamma \vdash M :_\downarrow A \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow M' \quad B[M'/x] \Downarrow B' \quad \Gamma \vdash N :_\downarrow B'}}{\Gamma \vdash (M, N) :_\downarrow \begin{bmatrix} x{:}A \\ B \end{bmatrix}}\,(\underline{\Sigma I})$$

$$\frac{\Gamma \vdash M :_\uparrow \begin{bmatrix} x{:}A \\ B \end{bmatrix}}{\Gamma \vdash \pi_1(M) :_\uparrow A}\,(\underline{\Sigma E}) \qquad \frac{\Gamma \vdash M :_\uparrow \begin{bmatrix} x{:}A \\ B \end{bmatrix} \quad B[\pi_1(M)/x] \Downarrow B'}{\Gamma \vdash \pi_2(M) :_\uparrow B'}\,(\underline{\Sigma E}) \qquad \frac{\Gamma \vdash M :_\uparrow (x{:}A) \to B \quad \Gamma \vdash N :_\downarrow A \quad B[N/x] \Downarrow B'}{\Gamma \vdash MN :_\uparrow B'}\,(\underline{\Pi E}) \qquad \text{ただし、} \mathsf{s}, \mathsf{s}_1, \mathsf{s}_2 \in \{\mathsf{type}, \mathsf{kind}\}$$

図 3: UDTT の型規則（抜粋）

うか型チェックを行った後、その証明木に対して @-Elimination を行うことで A の中の @ を取り除き、B の型チェックを行う際の型環境に加えている。これにより、文の意味表示中に複数の @ がある場合でも、より深い位置に存在する @ から順に解消していくことが可能となっている。

2.2 DTS の証明探索

DTS の証明探索は、@ 規則において型チェックの一部として呼び出されるよう定義されている。@ 規則上段の $\Gamma \vdash A'\ \mathit{true}$ が証明探索の呼び出される箇所である。

DTS のための証明探索アルゴリズムとして、[7] において以下のアルゴリズムが提案されている。

1. 証明探索を行うときの型環境とシグネチャの中から、Σ 型を持つ項を探す
2. 1 で見つかった項に投射を適用することで、Σ 型の項の第一要素・第二要素とその型を得る
3. 1,2 を型環境とシグネチャの中から Σ 型がなくなるまで繰り返す
4. 同じ型環境とシグネチャの中から、Π 型を持つ項を探す
5. 4 で見つかった項に関数適用できる項を 3 で得られた項の中から探し、関数適用した結果の項とその型を得る
6. 4,5 を型環境・シグネチャから Π 型がなくなるまで繰り返す
7. 1 から 6 の操作で得られた項に対応する型の中から、証明探索を行う型と一致する型を探し、その型に対応する項を返す

このアルゴリズムを用いることにより、(CHK) 規則、(CON) 規則、(VAR) 規則、(ΣE) 規則、(ΠE) 規則のみを特定の順序で用いることで証明探索が可能な型について、証明項を見つけることができる。

3 実装

DTS のための型チェック・証明探索アルゴリズムは [7] において実装されているが、[7] ではパーザとの接続はなされていなかった。そこで本研究では、CCG パーザ lightblue との接続が可能となるよう型チェック・証明探索アルゴリズムの再実装を行った。

パーザとの接続を考える上で [7] での実装と最も異なるのは、lightblue の出力の意味表示は de Bruijn index[5] を用いていることである。de Bruijn index とは、変数を変数名で言及するのではなく、0 以上の自然数を用いて表す記法である。つまり、自然数 k は k 番目に束縛された変数に対応する。de Bruijn index を用いると、$\begin{bmatrix} x{:}\mathbf{entity} \\ \mathbf{man}(x) \end{bmatrix}$ という項は $\begin{bmatrix} \mathbf{entity} \\ \mathbf{man}(0) \end{bmatrix}$ と書くことができる。

de Bruijn index を用いた DTS のための型チェック・証明探索アルゴリズムの実装を行う際に注意しなくてはならないのが、(ΠE) 規則や (ΣE) 規則などの代入操作を伴う規則である。たとえば (ΠE) 規則ならば、規則上段の $B[N/x]$ という部分で代入操作が行われている。de Bruijn index を用いる場合、代入の際に変数の対応関係が崩れないよう注意する必要がある。具体的には、代入の際に代入する項の 0 以上の index に 1 追加し、その項を代入し終えた項全体の項の 0 以上の index を 1 下げる必要がある。このことに注意すれば、de Bruijn index を用いた DTS のための型チェック・証明探索アルゴリズムは [7] と同様の方法で実装することができる。

4 おわりに

本研究では、依存型意味論 (DTS) のための型チェック・証明探索アルゴリズムについて考察し、その実装を行った。さらに、日本語 CCG パーザとの接続も行った。

ただし、自然数型や枚挙型を含む Martin-Löf 型理論全体の型チェック・証明探索をどのように実現するかについては、今後の課題としたい。

参考文献

[1] Daisuke Bekki. Representing anaphora with dependent types. In *Logical Aspects of Computational Linguistics*, pages 14–29. Springer, 2014.

[2] Daisuke Bekki and Ai Kawazoe. Implementing Variable Vectors in a CCG Parser. In Proceedings of LACL2016, C.Retore and S.Pogodalla (Eds), pp.52-67, Springer, 2016.

[3] Andres Löh, Conor McBride, and Wouter Swierstra. A tutorial implementation of a dependently typed lambda calculus. *Fundamenta informaticae*, 102(2):177–207, 2010.

[4] Per Martin-Löf. *Intuitionistic Type Theory*, volume 17. Italy: Bibliopolis, Naples, 1984. Sambin, Giovanni (ed.).

[5] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[6] 佐藤未歩, 戸次大介. 依存型意味論のための型チェックの実装に向けて. 言語処理学会第 22 回年次大会発表論文集, pp.761–764, 2016.

[7] 佐藤未歩, 戸次大介. 依存型意味論における型チェックの実装の試み. 第 30 回人工知能学会全国大会論文集, 北九州国際会議場, 2016/6/6–6/9.