1N2-2

依存型意味論における型チェックの実装の試み

Towards an implementation of type checking in Dependent Type Semantics

佐藤未歩 *1　戸次大介 *1*2
Miho Satoh　Daisuke Bekki

*1お茶の水女子大学大学院
Graduate School of Ochanomizu University

*2独立行政法人科学技術振興機構, CREST
CREST, Japan Science and Technology Agency

Dependent Type Semantics (DTS) is a proof-theoretic, compositional discourse semantics of natural language based on Dependent Type Theory (DTT), which gives a unified analysis of anaphora and presupposition including such cases as bridging anaphora. In Bekki (2015) and Sato and Bekki (2016), the anaphora resolution and presupposition binding process in DTS is reformulated by the combination of Underspecified Dependent Type Theory (UDTT), its type checking/proof search algorithm, and translation from UDTT to DTT (@-Elimination). In this paper, we implemented this process in a decidable and compositional manner and demonstrate a resolution of bridging anaphora.

1. はじめに

照応解析において一般によく行われるのは、文脈中に照応代名詞が存在したときに、その照応代名詞よりも前の文脈中から名詞的な表現を取り出し、それらの中から照応代名詞の先行詞を探す、という処理である。一方で、照応現象にはこのような単純な解析では扱うことのできない様々な例が存在することが知られている。たとえば、以下の 3 つの例文は先行詞が名詞的ではない照応現象の例である。

(1) a. More than three students came. They were drunk.
b. John buys a car. Bill checks the moter.
c. If Mary smokes, John knows that she does.

文 (1a) は E タイプ照応 [5] と呼ばれるもののうち、先行詞が数量詞である例である。照応代名詞 *they* は *more than three students* を先行詞とするが、単純に *they* を *more than three students* で置き換えた文である *More than three students were drunk.* は (1a) の 2 文目とは真理条件が異なる。文 (1b) は橋渡し照応 (bridging anaphora) [4] の例である。2 文目に現れている *the moter* は 1 文目の *a car* の一部を指すが、先行詞が明示的に存在していない例である。文 (1c) は前提現象 (presupposition) [6] の例である。(1c) の後件部において *John* が知っている内容は前件部の *Mary smokes* であるが、名詞的な表現ではない。

上の 3 つのような現象は理論言語学において古くから議論されており、たとえば前提現象については Stalnaker, Karttunen, Heim らによって研究されてきた。のちに van der Sandt [9] らにより前提現象は照応に還元されうることが示され、前提投射の問題は照応解決と統一的に定式化されている。

一方で、文 (1b) のような橋渡し照応の例は談話表示理論 (Discourse Representation Theory)[7] を代表とする従来の形式意味論の枠組みでは扱うことの難しい例として知られている（中村ら (2015)[13]）。橋渡し照応を解決するためには先行文脈で明示的に与えられた情報と、先行文脈に明示的には存在しない世界知識とを組み合わせて先行詞を推論しなくてはならないからである。

依存型意味論 (Dependent Type Semantics, 以下 DTS)[1] は、上記の橋渡し照応を含む様々な照応現象を統一的に扱うことのできる自然言語の意味論である。DTS は依存型理論 [8] に基づく証明論的意味論であり、照応解消の計算は文の意味表示に対する型チェックや証明探索の問題に帰着するという特徴がある。DTS において (1a) のような先行詞が数量詞である場合の照応は田中ら (2015)[12] において議論されている。また、(1b) のような橋渡し照応の例は中村ら (2015)[13] において、(1c) のような叙実動詞の前提の例については田中ら (2015)[11] においてそれぞれ議論されている。

文の意味表示に対する型チェックは Bekki and Satoh (2015)[3] において定式化・実装され、一定の成果を納めているものの、そこにはいくつかの問題点が存在した。それらを解決しうる新たな DTS の体系が Bekki (2015)[2] によって提案され、佐藤・戸次 (2016)[10] においてその型チェック・型推論アルゴリズムが定式化された。本論文では、[10] において定式化された新たな DTS のための型チェック・型推論アルゴリズムの実装し、このアルゴリズムによって橋渡し照応を含む様々な照応現象を自動的に計算することが可能であることを示す。

2. DTS における照応の計算

DTS では、未指定項 (underspecified term) によって照応代名詞・前提トリガーの意味を表示する。たとえば、文 (1b) の意味表示は以下 (図 1) のように与えられる。

$$\left[\begin{array}{l} v{:}\left[\begin{array}{l} u{:}\left[\begin{array}{l} x{:}\mathbf{entity} \\ \mathbf{car}(x) \end{array}\right] \\ \mathbf{buy}(\mathbf{j}, \pi_1(u)) \end{array}\right] \\ \mathbf{check}\left(\mathbf{b}, \pi_1(@ : \left[\begin{array}{l} y{:}\mathbf{entity} \\ \mathbf{moterOf}(y, @ : \mathbf{entity}) \end{array}\right])\right) \end{array}\right]$$

図 1: 例文 (1b) の意味表示

図 1 中の @ が未指定項であり、照応代名詞などの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。DTS において照応解決は @ を具体的な証明項に置き換えることに相当する。たとえば、図 1 の意味表示には 2 つの @ が含まれており、それぞれを具体的な証明項に置き換えて以下のような意味表示を得ることで照応解決となる。（w は世界知識を表す。）

$$\left[\begin{array}{l} v{:}\left[\begin{array}{l} u{:}\left[\begin{array}{l} x{:}\mathbf{entity} \\ \mathbf{car}(x) \end{array}\right] \\ \mathbf{buy}(\mathbf{j}, \pi_1(u)) \end{array}\right] \\ \mathbf{check}\,(\mathbf{b}, \pi_1(w(\pi_1(\pi_1(v)))(\pi_2(\pi_1(v))))) \end{array}\right]$$

@ を具体的な証明項に置き換えるために、DTS では以下の 3 つの操作を行う。

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. 具体的な証明項による @ の置き換え

連絡先: 佐藤未歩，お茶の水女子大学大学院人間文化創成科学研究科戸次研究室，東京都文京区大塚 2-1-1, satoh.miho@is.ocha.ac.jp

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

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

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

図 2: UDTT の型規則（抜粋）

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

$$\left[\!\!\left[\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \mathsf{s} \quad [\![\mathcal{D}_1]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma, x : A' \vdash M : B} \quad \overset{\mathcal{D}_2}{}}{\Gamma \vdash \lambda x.M : (x{:}A) \to B}\ (\underline{\Pi I}) \right]\!\!\right] = \frac{[\![\mathcal{D}_1]\!] \quad [\![\mathcal{D}_2]\!]}{\Gamma \vdash A' : \mathsf{s} \quad A' \Downarrow A'' \quad \Gamma, x : A'' \vdash M' : B'}\ (\Pi I) \qquad \left[\!\!\left[\frac{\overset{\mathcal{D}}{\Gamma \vdash M : \begin{bmatrix} x{:}A \\ B \end{bmatrix} \quad B[\pi_1(M)/x] \Downarrow B'}}{\Gamma \vdash \pi_2(M) : B'}\ (\underline{\Sigma E}) \right]\!\!\right] = \frac{[\![\mathcal{D}]\!]}{\Gamma \vdash M' : \begin{bmatrix} x{:}A' \\ B' \end{bmatrix} \quad B'[\pi_1(M')/x] \Downarrow B''}{\Gamma \vdash \pi_2(M) : B''}\ (\Sigma E)$$

$$\left[\!\!\left[\frac{\overset{\mathcal{D}_1}{\Gamma \vdash M : (x{:}A) \to B \quad \Gamma \vdash N : A \quad B[N/x] \Downarrow B'} \quad \overset{\mathcal{D}_2}{}}{\Gamma \vdash MN : B'}\ (\underline{\Pi E}) \right]\!\!\right] = \frac{[\![\mathcal{D}_1]\!] \quad [\![\mathcal{D}_2]\!]}{\Gamma \vdash M' : (x{:}A') \to B' \quad \Gamma \vdash N' : A' \quad B'[N'/x] \Downarrow B''}\ (\Pi E)$$

$$\left[\!\!\left[\frac{\overset{\mathcal{D}_1}{\Gamma \vdash A : \mathsf{s}_1 \quad [\![\mathcal{D}_1]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma, x : A' \vdash B : \mathsf{s}_2} \quad \overset{\mathcal{D}_2}{}}{\Gamma \vdash \begin{bmatrix} x{:}A \\ B \end{bmatrix} : \mathsf{s}_2}\ (\underline{\Sigma F}) \right]\!\!\right] = \frac{[\![\mathcal{D}_1]\!] \quad [\![\mathcal{D}_2]\!]}{\Gamma \vdash A' : \mathsf{s}_1 \quad A' \Downarrow A'' \quad \Gamma, x : A'' \vdash B' : \mathsf{s}_2}\ (\Sigma F) \qquad \left[\!\!\left[\frac{\overset{\mathcal{D}}{\Gamma \vdash M : \begin{bmatrix} x{:}A \\ B \end{bmatrix}}}{\Gamma \vdash \pi_1(M) : A}\ (\underline{\Sigma E}) \right]\!\!\right] = \frac{[\![\mathcal{D}]\!]}{\Gamma \vdash M' : \begin{bmatrix} x{:}A' \\ B' \end{bmatrix}}{\Gamma \vdash \pi_1(M') : A'}\ (\Sigma E)$$

$$\left[\!\!\left[\frac{\overset{\mathcal{D}_1}{\Gamma \vdash M : A \quad [\![\mathcal{D}_1]\!]^{\mathsf{tm}} \Downarrow M' \quad B[M'/x] \Downarrow B' \quad \Gamma \vdash N : B'} \quad \overset{\mathcal{D}_2}{}}{\Gamma \vdash (M, N) : \begin{bmatrix} x{:}A \\ B \end{bmatrix}}\ (\underline{\Sigma I}) \right]\!\!\right] = \frac{[\![\mathcal{D}_1]\!] \quad [\![\mathcal{D}_2]\!]}{\Gamma \vdash M' : A' \quad B'[M'/x] \Downarrow B'' \quad \Gamma \vdash N : B''}\ (\Sigma I) \qquad \text{ただし、} \mathsf{s}, \mathsf{s}_1, \mathsf{s}_2 \in \{\mathsf{type}, \mathsf{kind}\}$$

図 3: @-Elinination の定義（抜粋）

1 の型チェックによって文の意味表示中の @ の型を確かめ、2 の証明探索によって @ に対する具体的な証明項を見つけ、3 によって @ を具体的な証明項で置き換えた意味表示を得ることで照応解決となる。3 の操作を @-Elimination と呼ぶ。

Bekki (2015)[2] では、DTS の体系を通常の依存型理論の体系 (Dependent Type Theory, DTT) と @ を含む依存型理論の体系 (Underspecified Dependent Type Theory, UDTT) に分けて定義している。そのため、[10] では DTT の型チェック・型推論と UDTT の型チェック・型推論の両方を定式化している。UDTT の型規則（抜粋）を図 2 に示す。@ を具体的な証明項に置き換えるための 3 つの操作のうち、1 と 2 は UDTT における操作であり、3 は文の意味表示を UDTT での意味表示から DTT での意味表示に変換する操作として定義されている。また、2 の証明探索は UDTT の型規則の 1 つである (@) 規則において、1 の型チェックのなかで行うよう定式化されている。

$$\frac{\overset{\mathcal{D}}{\Gamma \vdash A :_{\downarrow} \mathsf{s} \quad [\![\mathcal{D}]\!]^{\mathsf{tm}} \Downarrow A' \quad \Gamma \vdash A'\ \mathit{true}}}{\Gamma \vdash (@ : A) :_{\uparrow} A}\ (\underline{@})$$

(@) 規則の右上の $\Gamma \vdash A'\ \mathit{true}$ の部分が証明探索を要する箇所である。また、UDTT の各型規則に現れる $[\![\mathcal{D}]\!]^{\mathsf{tm}}$ は @-Elimination を行う箇所である。@-Elimination の定義（抜粋）は図 3 に示す。

3. Haskell による実装

DTS のための型チェックアルゴリズムを実装するためには、DTT の体系と UDTT の体系の両方を実装しなくてはならない。DTT の体系の実装については [3] において定式化・実装を行ったものを一部変更することで与えられる。UDTT の体系の実装については、型チェック・型推論、証明探索、@-Elimination の 3 つを実装する必要がある。本研究では、プログラミング言語 Haskell を用いて実装を行った。

3.1 UDTT の型チェック

UDTT の型チェックは DTT と異なり、型チェックの途中で @-Elimination を行う。@-Elimination は UDTT の証明木に対する操作であるので、型チェックを行うと同時に証明木の情報を持ち歩くよう実装しなくてはならない。UDTT の証明木のノードには必ず以下のような式が現れる。

$$\Gamma \vdash M : A$$

この式のことを型判定という。このことから、UDTT の証明木を実装するには、型判定をノードに持つ木構造を定義すればよい。この考えに基づき、型判定に対応するデータ型として以下の `UJudgement` を定義した。

```
data UJudgement =
   UJudgement TUEnv UPreterm UPreterm
```

`UJudgement` は 3 つの要素を持ち、1 つ目の要素 `TUEnv` が $\Gamma \vdash M : A$ における環境 Γ、2 つ目の要素が項 M、3 つ目の要素が型 A にそれぞれ対応する。`UPreterm` は UDTT の項に対応するデータ型である。また、型判定をノードに持つ木構造に対応するデータ型として定義したのが、以下の `UTree a` である。

```
data UTree a =
     UEmpty
   | UCHK a (UTree a)
   | ···
   | ASP a (UTree a) (UTree a)
   | UPiF a (UTree a) (UTree a)
   | ···
   | USigE a (UTree a)
```

```
sig :: SUEnv
sig = [(UCon "w", UPi "x" (UCon "entity") (UPi " " (UApp (UCon "car") (UVar "x")) (USigma "y" (UCon "entity") (UApp (UCon "moterOf") (UPair (UVar "y") (UVar "x")))))),
(UCon "j", UCon "entity"), (UCon "b",UCon "entity"), (UCon "car", UPi " " (UCon "entity") UType),
(UCon "buy",UPi " " (USigma " " (UCon "entity") (UCon "entity")) UType), (UCon "check",UPi " " (USigma " " (UCon "entity") (UCon "entity")) UType),
(UCon "moterOf", UPi " " (USigma " " (UCon "entity") (UCon "entity")) UType), (UCon "entity", UType)]

preTerm :: UPreterm
preTerm = USigma "v" (USigma "u" (USigma "x" (UCon "entity") (UApp (UCon "car") (UVar "x"))) (UApp (UCon "buy") (UPair (UCon "j") (UProj One (UVar "u")))))
(UApp (UCon "check") (UPair (UCon "b") (UProj One (Asp (USigma "y" (UCon "entity") (UApp (UCon "moterOf") (UPair (UVar "y") (Asp (UCon "entity")))))))))


ansTree :: Maybe (UTree UJudgement)
ansTree = typeCheckU [] sig preTerm UType
```

図 4: テストプログラム

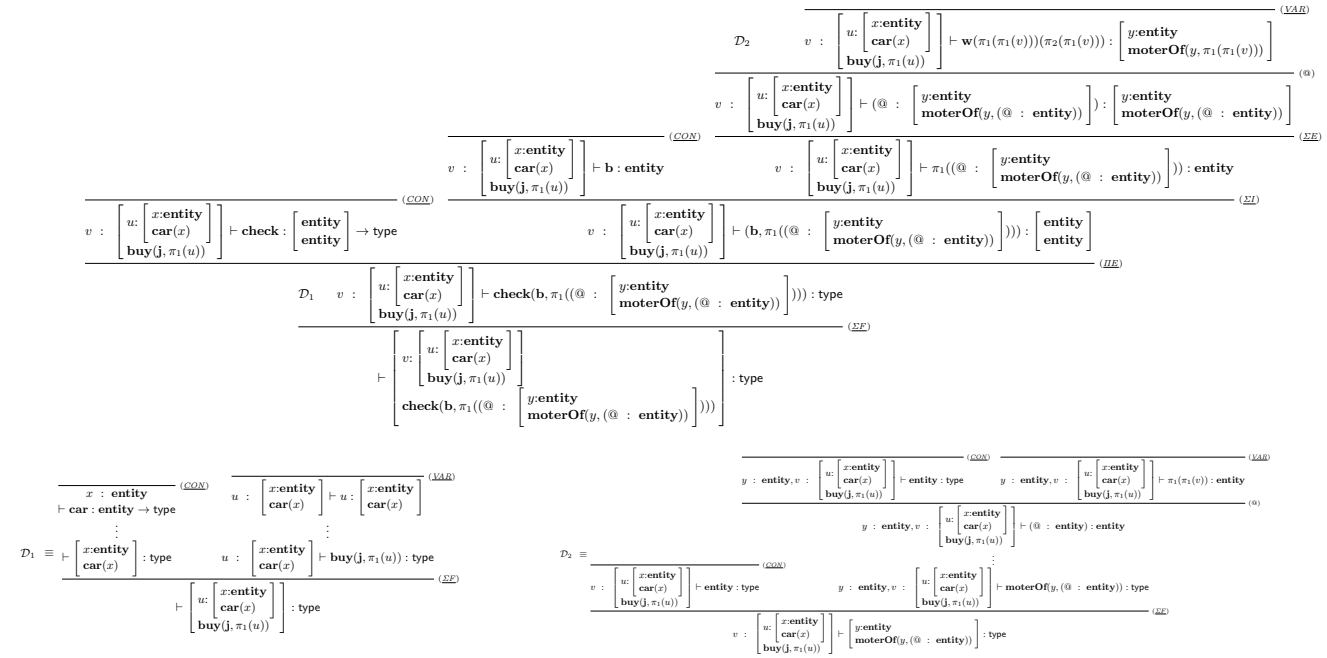

図 5: 型チェックの実行結果

UTree a の各値コンストラクタは、UDTT の各型規則に対応している。UTree a 型の値コンストラクタごとに引数の数が異なるのは、型規則によって葉の数が異なることに対応している。UTree a の型引数 a の部分に前述の UJudgement を与え、UTree UJudgement とすることで UDTT の型判定をノードに持つ木構造を表現することができる。(同様に DTT の証明木についても、Judgement と Tree a という 2 つのデータ型を定義することで表現しており、その内容は UDTT のものとほぼ同じである。)

UDTT の型チェックは typeCheckU と typeInferU という 2 つの関数を相互再帰的に定義することで実装しており、それぞれ型は以下のようになっている。

```
typeCheckU ::
 TUEnv -> SUEnv -> UPreterm -> UPreterm ->
 Maybe (UTree UJudgement)

typeInferU ::
 TUEnv -> SUEnv -> UPreterm ->
 Maybe (UTree UJudgement)
```

typeCheckU は UDTT の型チェック可能な項に対する型チェックを、typeInferU は UDTT の型推論可能な項に対する型推論を行う関数である。typeCheckU と typeInferU の両方に引数として渡している TUEnv と SUEnv は、それぞれ型環境とシグネチャを表現している。型環境は変数の型を保持する環境であり、変数名 (UPreterm) と対応する型 (UPreterm) のペアのリストである。型環境は型チェックの実行時に更新されていく。シグネチャは定数の型を保持する環境であり、定数名 (UPreterm) と対応する型 (UPreterm) のペアのリストである。定数と対応する型は型チェックの実行前に与えておくことが前提となるため、シグネチャは型チェックの実行時に更新されることはない。

関数 typeCheckU は型環境とシグネチャ、項、型を受け取り、与えられた項が型環境・シグネチャのもとで与えられた型を持つかどうか確かめ、与えられた型を持つ場合のみ、型チェックの結果を表す UDTT の証明木を返す。入力の項が入力の型を持たない場合、型チェックは失敗となるため、失敗つき計算を表現するために Maybe モナドを用いている。関数 typeInferU は型環境とシグネチャ、型推論を行う項を受け取り、型推論の結果判明した型を含む UDTT の証明木を返す。typeInferU も typeCheckU と同様の理由から Maybe モナドを用いており、失敗した場合にはどちらも Nothing が返る。

3.2 @-Elimination

@-Elimination は aspElim という関数を定義することで実装している。関数 aspElim の型は以下のようになっている。

```
aspElim ::
 (UTree UJudgement) -> Maybe (Tree Judgement)
```

関数 aspElim は、UDTT の証明木 (UTree UJudgement) を受け取って DTT の証明木 (Tree Judgement) を返す関数である。UDTT の証明木に aspElim を適用することで、証明探索によって判明した具体的な証明項によって @ を置き換えた DTT の証明木を得ることができる。

3.3 証明探索

UDTT において証明探索は (@) 規則中で呼び出されており、(@) 規則の右上の $\Gamma \vdash A'\ \ true$ の部分が証明探索を行っている箇所である。依存型理論において証明探索は一般に決定不能であるため、DTS においても証明探索の実装には注意が必要

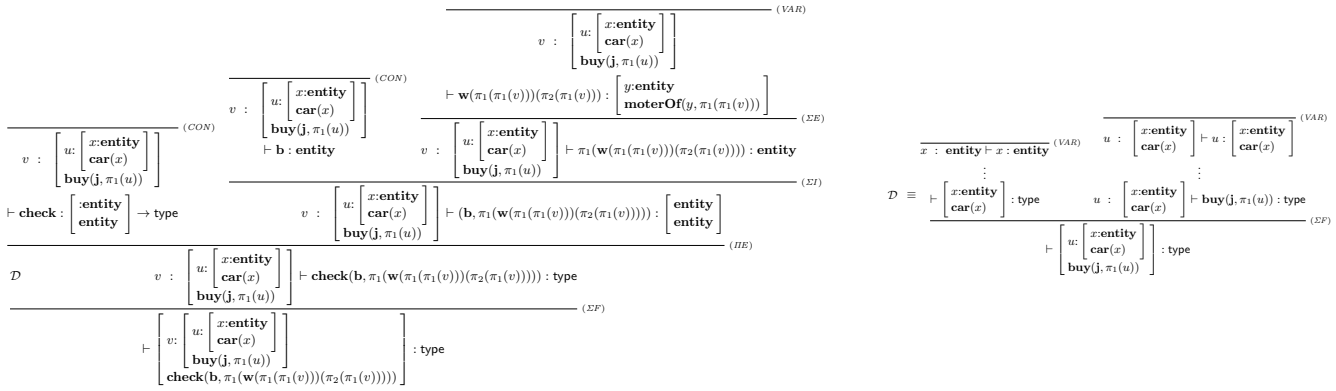

図 6: @-Elimination の実行結果

である。そこで、本研究では UDTT の証明探索のために以下のようなアルゴリズムを考え、実装した。

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・シグネチャから 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、あれば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

上のアルゴリズムにおける、1 から 3 の操作を行う関数が `dismantle`、4 から 6 の操作を行う関数が `execute` である。型はそれぞれ以下のように定義されている。

```
dismantle ::  TUEnv -> TUEnv -> TUEnv
execute ::  SUEnv -> TUEnv -> TUEnv -> TUEnv
```

関数 `dismantle` は上述 1,2,3 の操作を行い、結果として得られた全ての項と型のペアのリストを返す。関数 `execute` は上述 4,5,6 の操作を行い、結果として得られた項と型のペアのリストを返す。

実際に証明探索を行う際には、型環境には `dismantle`、シグネチャには `dismantle` と `execute` の両方を適用し、その結果得られた全ての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という操作を行っている。このアルゴリズムによる証明探索は限定的ではあるものの、Σ 型に対する投射や Π 型に対する関数適用など、基本的な操作で証明可能なものは扱うことができる。これにより、単純な照応現象だけでなく、橋渡し照応などの世界知識を必要とする照応現象や叙実動詞の前提など、様々な照応現象を計算することが可能であると考えられる。

3.4 テスト

この型チェックアルゴリズムのテストとして、図 4 のようなテストプログラムを作成した。このプログラムは、文 (1b) に対応する意味表示が型 type を持つかどうか型チェックを行い、さらにその結果得られた証明木に @-Elimination を適用したものである。文の意味表示に対する型チェックを行うことで意味表示中に現れる 2 つの @ の型を推論し、またその型に対する証明探索を行うことで具体的な証明項を得ることができている。また、@-Elimination を行うことで @ を具体例な証明項で置き換えた証明木を得ることができる。テストプログラムにおいて、型チェックと @-Elimination の実行結果はどちらも Haskell のプログラムとして出力されるので、本論文ではそれを Tex コードに変換し、証明木として読める形式で出力している。型チェックを行った結果の証明木が図 5、それに @-Elimination を行った結果の証明木が図 6 である。これらはどちらも理論的予測と一致しており、橋渡し照応の解決を自動で行うことができたことを示している。

4. おわりに

本研究では、Bekki (2015)[2]、佐藤・戸次 (2016)[10] において再定式化された依存型意味論 (DTS) のための型チェック・型推論アルゴリズムを、プログラミング言語 Haskell を用いて実装した。これにより、bridging を含む様々な照応現象を統一的に計算可能になったと言える。

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. Anaphora and presuppositions in dependent type semantics. A talk at Dynamic Semantics: Modern Type Theoretic and Category Theoretic Approaches, Ohio State University, United States, 24-25 October, 2015.

[3] Daisuke Bekki and Miho Satoh. Calculating projections via type checking. *Proceedings of TYTLES*, 2015.

[4] Herbert H Clark. Bridging. In *Proceedings of the 1975 workshop on Theoretical issues in natural language processing*, pages 169–174. Linguistics, 1975.

[5] Gareth Evans. Pronouns. *Linguistic Inquiry*, 11(2):337–362, 1980.

[6] Irene Heim. On the projection problem for presuppositions. *Formal Semantics–The Essential Readings*, pages 249–260, 1983.

[7] Hans Kamp and Uwe. Reyle. *From Discourse to Logic*. Kluwer Academic Publishers, 1993.

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

[9] Rob A. van der Sandt. Presupposition projection as anaphora resolution. *Journal of Semantics*, 9(4):333–377, 1992.

[10] 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.

[11] 田中リベカ, 峯島宏次, 戸次大介. 依存型意味論における叙実動詞の意味記述の試み. 第 29 回人工知能学会全国大会論文集, はこだて未来大学, 2015/5/30–6/2.

[12] 田中リベカ, 峯島宏次, 戸次大介. 依存型意味論による複数照応の分析. 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.

[13] 中村絢子, 峯島宏次, 戸次大介. オントロジーを用いた自然言語の推論に向けて. 言語処理学会第 21 回年次大会発表論文集, pp.309–312, 2015.