DTS の部分体系のための 定理自動証明器の実装に向けて

大洞日音
お茶の水女子大学
daido. hinari@is.ocha.ac.jp

戸次大介
お茶の水女子大学
bekki@is.ocha.ac.jp

1 自然言語推論のための定理証明器

理論言語学の目的の一つは言語現象を説明することであり，検証可能性の確保が重要であるが，大きなデータについて理論的予測を手計算で確認するのは非常にコストが高い．そのため，近年は形式統語論・意味論に基づく統語解析器・意味計算を，定理証明器と組み合わせることにより，テキストから推論までのパイプライン処理を自動で行うというアプローチが盛んに研究されている．Boxer/Nutcracker [1] では意味論モジュールとして Discourse Representation Theory (DRT) を採用し，組み合わせ範疇文法 (CCG) による統語解析器と，一階述語論理のための定理自動証明器が結合されたが，一階述語論理は照応・前提などの動的な現象を構成的に扱うことができない．この問題を克服するアプローチとして高階論理のための自然言語推論システムである LangPro[2] や ccg2lambda[3] が開発されているが，高階論理のための汎用的な定理自動証明器は未だ存在しない．このような状況で，言語推論に特化した，高階論理のための定理自動証明器の開発は大きな意義を持つ．

依存型意味論 (DTS) [4] は自然言語のための証明論的意味論であり，高階論理の一種である依存型理論 (DTT) を基礎としている．DTS は様々な動的な意味現象をカバーした合成的意味論でもあり，DTS を用いた自然言語推論パイプラインの確立は，形式意味論に基づく深い意味解析の実現にとって非常に望ましいものである．しかし，DTT における証明探索が決定不能であることから，DTS の完全な体系を採用した定理自動証明器の実装は難しく，先行研究は Piwek [5] や Bekki and Sato [6] など，少数の例外を除いて存在しない．しかし，完全ではなくても自然言語理解のために用いる範囲の DTT をカバーする定理証明器があれば，自然言語推論タスクに用いるには十分である．そこで本研究では，DTS の断片のための定理自動証明器を新規に開発し，定理証明器の評価用ライブラリである Thousands of Problems for Theorem Provers (TPTP) [1)] を用いて定量的な評価を行い，ある程度の推論を網羅できることを示した．

$$\left[\begin{array}{l} v : \left[\begin{array}{l} u : \left[\begin{array}{l} x : \mathbf{entity} \\ \mathbf{man}(x) \end{array}\right] \\ \mathbf{enter}(\pi_1 u) \end{array}\right] \\ \mathbf{whistle}(\pi_1 \pi_1 v) \end{array}\right]$$

図 1 DTS の例

$$\exists x(\mathbf{man}(x) \wedge \mathbf{enter}(x)) \wedge \mathbf{whistle}(x)$$

図 2 一階述語論理の例

2 依存型意味論 (DTS)

高階型理論の一種である DTS は，関数型と直積型の一般化として Π 型と Σ 型を用いる．Π 型は $(x : A) \to B$ という形で表記し，後件の型 B は前件の型 A をもつ項に依存する．Σ 型は $(x : A) \times B$ と表記し，Π 型と同様に B が型 A をもつ項に依存する．この性質から DTS は前提・照応など自然言語における複雑な現象を統一的に扱うことができる．図 1 は “A man enters. He whistles.” から合成的に計算された DTS の意味表現である．一方，この文を語の意味から合成的に計算して一階述語論理で表記すると図 2 のようになるのが自然であるが，これは存在量化子のスコープが適切ではない．このように，自然言語の意味計算には一階述語論理の記述力を越える現象が多々存在するが，高階論理を用いた DTS では自然に扱うことができる．

DTS の定理証明は型理論における項の探索に対応している．型理論では $\Gamma \vdash a : A$ という judgment を用いて文脈 Γ において項 a の型が A であることを表す．一方，論理的観点からこの judgment をみると，これは前提 Γ のもとで命題 A の証明が a であ

1） http://www.tptp.org/

る，という意味である．このことから，a は証明項とも呼ばれ，証明項を探索することは証明探索と呼ばれる．

$$t : \left[\begin{array}{l} u : \left[\begin{array}{l} v : \left[\begin{array}{l} x : \textbf{entity} \\ \textbf{girl}(x) \end{array}\right] \\ \left[\begin{array}{l} y : \textbf{entity} \\ \textbf{thesis}(y) \end{array}\right] \end{array}\right] \\ \textbf{write}(\pi_1\pi_1 u, \pi_1\pi_2 u) \end{array}\right] \vdash ? : \left[\begin{array}{l} x : \textbf{entity} \\ \textbf{girl}(x) \end{array}\right]$$

図 3　証明探索の例

図 3 で “?” の部分に当てはまる項を探索すると $(\pi_1\pi_1\pi_1 t, \pi_2\pi_1\pi_1 t)$ という証明項を得ることができ，これをデコードすることで証明探索においてどのような規則をどのような順番で適用したかという情報を得ることができる．

3　実装

本研究で実装した定理証明器の構造を図 6 に示す．

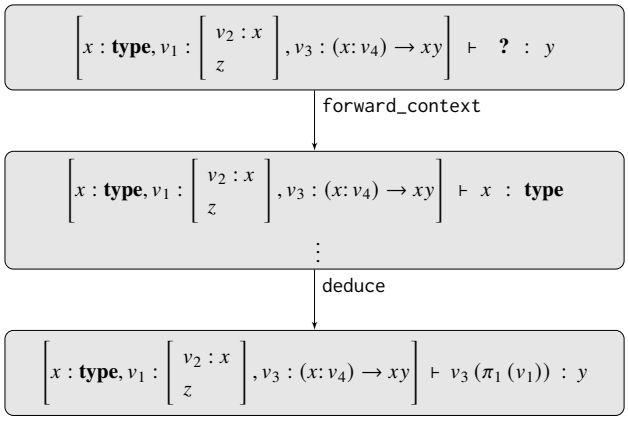

図 6　定理証明器の構造

アルゴリズムは Alligator [5] において提案されたものの改良である．`forward_context` では前向き推論を，`deduce` では後ろ向き推論を行っている．前向き推論では，前提と結論が与えられたとき，前提に規則を繰り返し適用することで上から下へと自然演繹形式の証明木を構築し，前提から導きうる命題を蓄積する．後ろ向き推論では，まず結論に規則を適用して証明のために必要なゴールを割り出し，下から上に証明木を構築していく形で推論を行う．`forward_context` は引数として前提を受け取り，前向き推論の結果として得られた judgment のリストを返す．`forward_context` で使用する規則の一部を図 4 に示す．`deduce` は図 5 にあるような規則を用いて後ろ向き推論を行い，judgment のリストを返す．`deduce` は無限再帰に陥る可能性があるため，探索の深さや，かかる時間に制限を置くことで，計算の制御を行っている．

4　評価

実装にはプログラミング言語 Haskell を用い，実験環境には産総研の AI 橋渡しクラウド (ABCI) [2] を用いた．定理自動証明器の評価のための包括的なライブラリである TPTP から，Alligator でカバーされていない等号や四則演算の問題を除いたデータセットによって評価を行った．TPTP ライブラリに収録されている問題は図 7 のような形式である．

```
%----------------------------------------------
%File     : SYN001+1 : TPTP v7.3.0. Released v2.0.0.
%Status   : Theorem
%Syntax   : Number of predicates  :   1(  1
    propositional; 0-0 arity)
%Comments :
%----------------------------------------------
fof(pel2,conjecture,( ~~p <=> p )).
```

図 7　TPTP に収録されている二重否定律の問題

推論の結果と各評価ラベルの関係は，図 8 のとおりである．

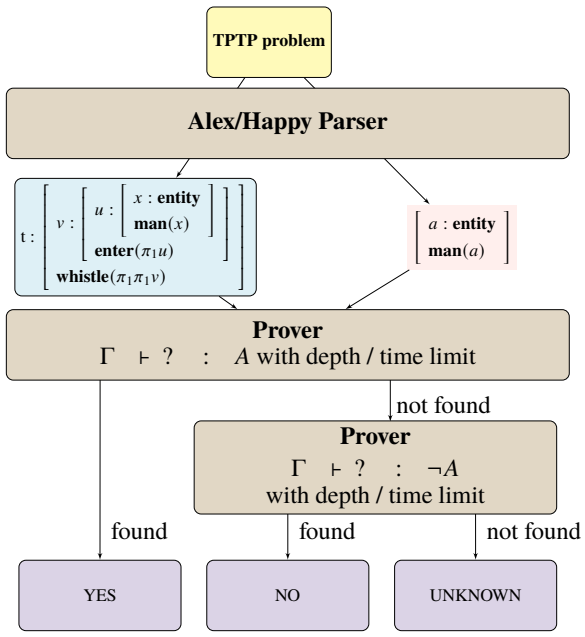

図 8　評価の形式

4.1　構文解析器：

Haskell の Alex/Happy library [7, 8] を用いて TPTP フォーマットから Haskell のデータ型への変換を行う Alex/Happy Parser を実装した．このデータ型を中間言語として，読み込んだ TPTP 形式の問題を，本

2）　https://abci.ai/

$$\frac{\Gamma \vdash_F z : \left[\begin{array}{l} x : A \\ B \end{array}\right]}{\Gamma \vdash_F \pi_1(z) : A}$$

(Σ-elim 1-1)

$$\frac{\Gamma \vdash_F z : \left[\begin{array}{l} x : A \\ B \end{array}\right]}{\Gamma \vdash_F \pi_2(z) : B[\pi_1(z)/x]}$$

(Σ-elim 1-2)

$$\frac{\Gamma \vdash_F z : \Gamma_1 \Rightarrow \left[\begin{array}{l} x : A \\ B \end{array}\right]}{\Gamma \vdash_F \lambda x_1 : m_1 \ldots x_n : m_n \,.\, \pi_1(z)(x_1) \ldots (x_n) : \Gamma_1 \Rightarrow A}$$

(Σ-elim 2-1)

$$\frac{\Gamma \vdash_F z : \Gamma_1 \Rightarrow \left[\begin{array}{l} x : A \\ B \end{array}\right]}{\Gamma \vdash_F \lambda x_1 : m_1 \ldots x_n : m_n \,.\, \pi_2(z)(x_1) \ldots (x_n) : \Gamma_1 \Rightarrow B'}$$

(Σ-elim 2-2)

$$\Gamma_1 = [x_1 : m_1, \ldots, a_n : x_n]$$
$$B' = B[\pi_1(z)/x]$$

図 4　forward_context の規則 (一部)

$$\frac{}{\epsilon \vdash s : s'}$$

(axiom)

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[x/a] \quad \Gamma \vdash (x{:}x) \to AB : s}{\Gamma \vdash (a, b) : \left[\begin{array}{l} x : A \\ B \end{array}\right]}$$

(Σ-intro)

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\left[\begin{array}{l} x : A \\ B \end{array}\right]) : s_2}$$

(Σ-form)

$$\frac{\Gamma, \Gamma_1 \vdash b : B \quad \Gamma \vdash \Gamma_1 \Rightarrow B : s}{\Gamma \vdash (\lambda x_1 : m_1\ x_2 : m_2 \ldots x_n : m_n \,.\, b) : \Gamma_1 \Rightarrow B}$$

(Π-intro)

$$\frac{\Gamma \vdash F : [x : A] \Rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash F(a) : B[a/x]}$$

(Π-elim)

$$\frac{\Gamma \vdash m_i : s_i \quad \Gamma, \Gamma_1 \vdash B : s}{\Gamma \vdash \Gamma_1 \Rightarrow B : s}$$

(Π-form 1-1)

$$s,\ s_i \in [type,\ kind],\ 1 \leqq i \leqq n$$

図 5　deduce の規則 (一部)

研究の定理証明器が対応している Haskell データ形式や Alligator が対応している Prolog データ形式へ翻訳する．

4.2　証明探索：

本研究の定理証明器は，与えられた文脈 Γ と命題 A について，$\Gamma \vdash ? : A$ という問題の証明項が見つかったときは YES ラベルを，$\Gamma \vdash ? : \neg A$ という問題の証明項が見つかったときは NO ラベルを返す．証明探索が停止せず，証明項が見つからなかった場合は UNK ラベルを返すが，このとき必ずしも命題が偽であるとは言えない点に注意する．

4.3　結果：

探索の深さ制限を 8 と 9 に設定した際の結果を表 1 に示す．この結果から，accuracy は探索の深さ制限に依存することが考えられる[3)]．表 2 で，YES ラベルと NO ラベル両方について precision は 1 であることから，推論での規則の適用は適切であると考えられる．一方，recall はどちらについても低く，特に NO ラベルについての recall は YES ラベルと比べ

表 1　探索の深さを 9/8 としたときの結果

	YES	NO	UNK
YES	46/45	0/0	0/0
NO	0/0	2/2	0/0
UNK	232/233	701/701	224/224

表 2　探索の深さを 9 としたときの結果詳細

	Prec	Rec	F1	Supp
YES	1.000	0.165	0.284	278
NO	1.000	0.003	0.006	703
UNK	0.194	1.000	0.324	224
Macro	0.731	0.389	0.205	1205
Micro	0.226	0.226	0.226	1205

3)　深さ制限を 7，6，5 とし，時間制限をなくして行った追加実験では，答えが YES である問題についての accuracy スコアは 0.104，0.050，0.025 であることが確認された．

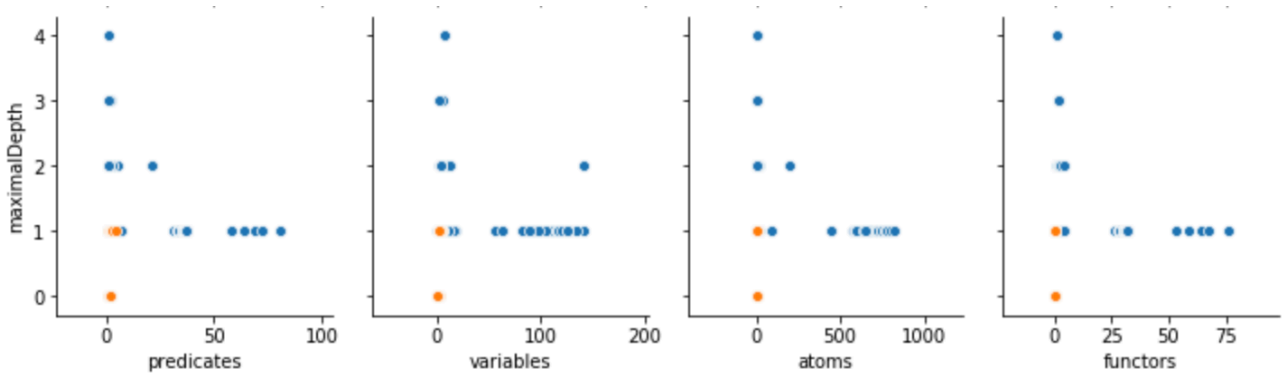

図 9　問題の複雑さと証明探索の結果の関係

ても低い．これは，評価の過程で NO ラベルの証明の際，命題に否定記号 (¬) を追加しており，命題の構造が複雑になる分，深さ制限を超過しやすくなるためと考えられる．図 9 はラベルが YES である問題について，問題の複雑さと証明探索の結果の関係を示すペアプロットである．オレンジのドットは証明探索の結果出力されたラベルが YES であったことを，青のドットは UNK であったことを表す．ペアプロットの y 軸は命題の深さを，x 軸は述語，変数，アトム，ファンクタの数を表しており，いずれも高ければ高いほど問題の複雑さは増す．この結果から，今回実装した定理証明器は x と y の値が小さい単純な問題については正確に計算を行えていることが示された．深さ制限を緩めると計算量が指数関数的に増加する一方，より多くの問題を解けるようになることが期待される．

本研究の定理証明器は，古典論理と直観主義論理の双方に対応しており，TPTP の問題の中でもパースの法則 $(p \Rightarrow q) \Rightarrow p) \Rightarrow p$ や対偶律 $(p \Rightarrow q) \Longleftrightarrow (\neg q \Rightarrow \neg p)$，排中律 $p \vee \neg p$ など，推論において重要な役割を果たす定理についてもカバーすることを確認した．

5　おわりに

本研究では DTS の部分体系（ΠΣ 断片）について証明探索アルゴリズムを実装し，TPTP ライブラリを用いて評価を行った．その結果，十分な範囲の定理について適切に自動計算を行うことを確認した．

今後は自然言語推論で用いられる，等号や算術についても扱えるようアルゴリズムを拡張していく．また，日本語 CCG パーザ lightblue[9] と組み合わせることによって，自然言語推論のためのパイプラインを構築し，DTS に基づく検証可能な意味計算システムを実現することが今後の課題である．

謝辞

本研究の一部は，JSPS 科研費 JP18H03284，および JST CREST JPMJCR20D2 の助成を受けたものである．また，評価の際は産総研の AI 橋渡しクラウド (ABCI) を利用した．

参考文献

[1] Johan Bos. Wide-coverage semantic analysis with Boxer. In *Semantics in Text Processing. STEP 2008 Conference Proceedings*, pp. 277–286, 2008.

[2] Lasha Abzianidze. LangPro: Natural language theorem prover. In *Proceedings of EMNLP2017 System Demonstrations*, pp. 115–120, Copenhagen, Denmark, September 2017.

[3] Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. ccg2lambda: A Compositional Semantics System. In *Proceedings of ACL 2016 System Demonstrations*, pp. 85–90, 2016.

[4] Daisuke Bekki and Koji Mineshima. Context-passing and underspecification in Dependent Type Semantics. In *Modern Perspectives in Type Theoretical Semantics*, pp. 11–41. Springer, 2017.

[5] Paul Piwek. The alligator theorem prover for dependent type systems: Description and proof sample. In *Proceedings of the Inference in Computational Semantics Workshop*, pp. 157–163, 2006.

[6] Daisuke Bekki and Miho Satoh. Calculating projections via type checking. In *TYTLES: Types Theory and Lexical Semantics, Robin Cooper, Christian Retore (eds.)*, pp. 62–68, 2015.

[7] alex: Alex is a tool for generating lexical analysers in Haskell. https://hackage.haskell.org/package/alex/.

[8] happy: Happy is a parser generator for Haskell. https://hackage.haskell.org/package/happy.

[9] Daisuke Bekki and Ai Kawazoe. Implementing variable vectors in a ccg parser. In *LACL2016, Nancy, France, C.Retore and S.Pogodalla (Eds)*, pp. 52–67, Heiderburg, 2016.