National Institute of
Advanced Industrial Science
and Technology
AIST

AIST-PS-2008-007

再帰的定義を可能にする述語論理の証明支援系上の実装 (Preliminary Version)

矢田部 俊介

独立行政法人 産業技術総合研究所
システム検証研究センター

算譜科学研究速報
Programming Science
Technical Report

Research Center
for Verification and Semantics

独立行政法人 産業技術総合研究所 システム検証研究センター 発行
Published by Research Center for Verification and Semantics
National Institute of Advanced Industrial Science and Technology

再帰的定義を可能にする述語論理の証明支援系上の実装 (Preliminary Version)

矢田部 俊介
独立行政法人 産業技術総合研究所
システム検証研究センター
shunsuke.yatabe@aist.go.jp

平成 20 年 3 月 13 日

概要

型付けを持つ証明支援系において、任意有限個の変数を持つ述語を含み、さらに量化子による変数の束縛が可能となる形式的体系（データ型やデータ構造）を定義することが困難であることはよく知られている。さらに、束縛された変数への具体的な値を代入する操作を証明支援系が解釈することにも困難が伴う。そのため、量化子を含む論理式によって関数を再帰的に定義しようとして、たとえそれが人間の目には十全な定義であっても、証明支援系はその関数の停止判定に失敗することが起こる。

本論文では、Martin-Löf の直観主義的型理論に基づいた厳格な型付けを持つ証明支援系 Agda 上で以上の点を解決した述語論理の実装例を紹介する。この例においては、多変数述語の実装は、依存型理論 (dependent type theory) を使用した。すなわち、自由変数の個数と、そのうちの束縛された変数の個数を、同時に型に情報として含むように、多変数述語のデータ型を定義した。また、束縛された変数の個数の上での帰納法により、知識様相述語論理の充足関係を関数として定義した。このことにより、関数定義が明示的に有限回で停止することを証明支援系に理解させることができ、関数の停止判定に成功した。

1 はじめに

本論文の題材は、Martin-Löf の直観主義的型理論 [4] に基づいた厳格な型付けを持つ証明支援系 Agda[5] 上での、知識様相論理 [7] の意味論（充足関数）の実装に関する技術的問題からとられた。目的の充足関数は、述語論理の上の証明可能性関係によって再帰的に定義されており、この関数の実装には、任意有限個の変数を持つ述語を持つ述語論理と、再帰的定義を可能とする様相論理の論理式のデータ型の定義が不可欠であった。

型付けを持つ証明支援系において、任意有限個の変数を持つ述語を含み、さらに量化子による束縛が可能となる形式的体系（データ型やデータ構造）を定義することが困難であることはよく知られている。そのため、例えば証明支援系 Agda は Martin-Löf の直観主義的型理論に基づいた厳格な型付けを持つが、その上で実装された述語論理の体系は、これまで述語として 1 変数述語のみしか持っていなかった [6]。そして、1 変数のやり方を一般化した（自然な）方法では、多変数述語の型を再帰的に定義していくことになるため、Agda の型検査に合格することができない。

また、束縛された変数への具体的な値を代入する操作を証明支援系が解釈することにも困難が伴う。Agda は依存型理論 (dependent type theory) に基づく証明支援系であるにもかかわらず、関数の原始再帰法による定義だけでなく、一般の再帰的定義 (general recursion) も許すことが特徴である [1]。そのため、関数などを

定義する際には、型の判定だけではなく、定義が本当に停止するかどうかも判定する仕様になっている。さて、代入操作は人間には意味が明確であっても、証明支援系にその意味を教えることは難しい。今回のように、一般的な再帰法を使用して量化子を含む論理式によって関数を定義しようする場合、関数定義に代入操作が含まれていると、たとえそれが人間の目には十全な定義であっても、証明支援系がその関数定義の停止判定に失敗することも起こりうる。

本論文では、Agda で以上の点をどのように解決したかを報告する。この実装例では、Agda 上で任意有限個の変数を持つ述語を備えた述語論理をデータ型としてまず定義し、その上で束縛された変数を含む論理式の上での再帰的定義によって、知識様相述語論理の体系の充足関数を定義した。

まず、述語論理の多変数述語は、ベクトルを使用し定義した。また、各種ベクトルの操作を定義することで、述語論理の意味論を定義することができた。

次に、様相論理の多変数述語を含む論理式のデータ型の実装は、依存型理論の特性を使用した。すなわち、自由変数の個数だけでなく、そのうちの束縛された変数の個数を同時に型に情報として含むように、依存型として様相論理の論理式のデータ型を定義した。最後に、知識様相述語論理の充足関数の定義には、束縛された変数の個数の上での帰納法を使用した。これは、束縛変数の個数の情報がデータ型からわかるようになったことにより可能になった。このことにより、関数定義が明示的に有限回で停止することを証明支援系に理解させることができ、関数の停止判定に成功した。

束縛された変数を持つデータ構造において、再帰的に関数を定義する際の困難は、Agda だけでなく、すべての型付けを持つ言語に共通している。その点で、今回の実装例は、他の証明支援系においても実装の際の参考になるものと期待される。

2 Agda と、その上の再帰的定義の持つ困難

今回の事例では、竹内 [7] で導入された知識様相論理の体系を Agda 上に実装することを目的とした。竹内の体系は、その意味論が任意有限個の変数を持つ述語を備えるような述語論理上の証明可能性によって、論理式の形によって再帰的に定義されるため、まず述語論理とその意味論を Agda 上で定義し、その枠組みの中で知識様相論理とその意味論を定義する。この章では、Agda および知識様相論理の体系を紹介し、また、一見うまくいきそうなそれらのテスト実装例を紹介する。

2.1 定理証明支援系 Agda

本節では、フレームワークとなる定理証明支援系 Agda と、その上での論理の実装法を紹介する。

2.1.1 Agda について

Agda では、Martin-Löf の直観主義的型理論に基づいた証明支援系である [4][5]。そこでは、命題は集合と同一視される。従って、p が命題であるとき、今後は $p : \mathbf{Set}$ と表記する。

Agda の備える直観主義型理論によって、述語論理のかなりの部分（命題論理をすべてと、述語論理の一部分）をそのまま定義することができる[*1]。ここでは述語論理における論理的帰結関係と全称量化子の取り扱いのみを紹介しよう。命題（集合）A, B に関して、$A \to B$ は、集合 A から B への関数として定義される。ま

[*1] Agda においては、純構文論的に述語論理を実装しているというよりも、BHK 解釈に基づくその意味論を、型理論の言葉で記述していると考えたほうがよい。

た、全称量化子こと「任意の $x : D$ に関して px」（通常の述語論理では $(\forall x \in D)p(x)$ と書くことが多い）は、$(x : D) \to px$ と、依存型として定義される。

2.1.2 一般の再帰法と停止判定の重要性

Agda の特徴の一つは、依存型理論に基づいていながら、一方で、関数の原始再帰法による定義だけでなく、一般の再帰法 (general recursion) も許すことにある。

「依存型理論では（原始再帰法以外の）一般の再帰法は使用できない」と言われることが多い。これは神話ではあるものの、確かに依存型理論において一般の再帰法は問題をもたらすことがある [1]。その例として、以下の依存型の判断を考えてみよう。

$$\frac{f : \forall x : S \to T[x], \quad s : S}{fs : T[s]}$$

このとき、関数 f の領域 (domain) の型と引数 (argument) s の型が一致している必要がある。しかし、例えば S として自然数をとったとき、x に関数の計算結果を入れることも可能となる。したがって $f(2+2)$ は $f(4)$ と型が一致し、また $T[2+2]$ は $T[4]$ と型が一致する必要がある。そして、この計算が停止しない場合、証明支援系は型付けに失敗する。

このため、例えば Coq では、一般の再帰法による関数定義は許可されない。このことは、証明系の論理的な健全さを保障するが、一方で関数定義が非常に難しくなり、プログラマーへの負担が大きくなる。

一方、Agda では一般の再帰法による定義を許す。その代わり、型検査の終了後に、別の検査「停止判定」を行う。これは、その関数の型の判定に計算が含まれている場合、その計算が停止するかどうかを、何らかの “cut-off mechanism” を使って判定する。したがって、Agda で関数を（一般の再帰法を使用して）定義する場合、この停止判定にパスするかどうかが最大の問題となる。

2.2 実装の目的と要求仕様

本節では、まず目的の体系を実装するために必要な述語論理の体系を紹介する。次に、知識様相論理の体系を紹介する。

2.2.1 メタ言語：述語論理

今回は、対象言語の様相論理の意味論を定義するためのメタ言語として、述語論理を Agda に実装する。Agda では、前述のように述語論理のある程度の部分をそのまま展開することができるが、一方で、任意の自然数 n に対して n 変数の述語を表現するデータ型を用意していない。従って、述語論理の完全な実装のために、直観主義型理論の枠内で、任意有限個の述語のデータ型を定義し、またそのデータ型（対象言語）の意味論を、直観主義型理論によって構成する必要がある。

また、その際には、変数の取り扱いに関し、今回の実装における要求仕様は以下の二点を求めていた。

- Agda の提供する変数の取り扱い機構をそのまま使用する（shallow embedding）。
- 代入操作等は、証明を行う際にパターンマッチを使用したいため、データ型のコンストラクターとして定義する。

2.2.2 対象言語：知識様相論理

さて、竹内 [7] の知識様相論理をまず紹介しよう。個体観念が共有されていても、各個人は個体同定を誤る事が起こりうる（つまり、誰がその部屋にいたかのリストは完璧でも、宝石を盗んだ犯人を間違えることは起こりうる）。この知識様相論理の体系は、個体同定に関する誤った情報を含む、伝聞に関する知識を表現する様相論理の体系である。しかし、個体同定に誤りを含む情報であっても、文脈に関する知識と照合すれば、その情報は決して情報量がゼロなのではなく、誤った情報から正しい結論を引き出すことも可能である（一つ一つは間違いを含む証言でも、突き詰めていけば、真犯人を推理できる）。この体系において、文脈は $\langle \vec{t}, \varphi, \psi \rangle$ で表現される。個体の領域を D とおく。$\bar{t}$ は共有された個体のリストで、$\bar{t} = (t_0, \cdots, t_{n-1})$ としたとき、任意の $i \leq n-1$ に対して $t_i : D$ となる。φ こと誤った情報を含みうる話し手の情報を「透過文脈」と呼び、聞き手の推理した正しいと思われる情報 ψ を「不透過文脈」と呼ぶ。

注意だが、この体系の構文論について、メタ言語の構文論と区別するため、この論文中では全称量化子 **forall**、論理的帰結関係 $\Rightarrow$、などの記号で表現する。メタ言語となる述語論理の記号は、$\forall$、$\rightarrow$ などで表現する。また、この体系の命題は、集合 p に対してコンストラクター **trans** を適用した $\mathbf{trans}\, p$ の形をしたものとして定義される（詳細は省く）。

この体系の特徴は、述語論理上、証明可能性関係を使って意味論上の充足関係を定義することにある。この論文では、実装の上で最大の問題となった、全称量化子（**forall**） のケースだけ紹介する[*2]。フレッシュな変数 y に関して、定義となる[*3]。

$$
\begin{aligned}
\langle \bar{s}, \varphi, \psi \rangle \models & (\mathbf{forall}\, x) P(x) \\
\Longleftrightarrow & \langle \bar{s}, \varphi \wedge ((\exists y) Q(y) \rightarrow Q), \\
& \quad \psi \wedge ((\exists y) R(y) \rightarrow R) \rangle \models P(y) \\
& \quad \text{が任意の } Q(x), R(x) \text{ に成立する}
\end{aligned}
$$

注意として、$\models (\mathbf{forall}\, x) P(x)$ の定義は、束縛変数 x にフレッシュな新しい変数 y を代入するため、難しいケース $\cdots \models (\mathbf{forall}\, x) P(x)$ を簡単なケース $\cdots \models P(y)$ に還元している。そのため、人間の目には、この再帰法が有限回で停止するのは明らかである（2.4.2 節で後述するとおり、この点が証明支援系にとっての問題となる）。なお、透過文脈・不透過文脈が直接に関係するのは否定文などのケースの処理であり、本論文の内容とは関係しない。

この論理は、個体同定に失敗した場合でも正しい結論を導くことができるため、ネットワーク上の認証の安全性の証明などに応用することが考えられる。

2.2.3 ポイント

ポイントになるのは、以下の二点が要求されるということである。

- 述語論理の実装：任意有限個の変数をもつ述語を、変数を束縛する機構（量化子・個体記号の変数への代入操作）を持つデータ型として定義すること。

[*2] このとき、$\langle \bar{s}, \varphi, \psi \rangle \models (\mathbf{forall}\, x) P(x)$ の直感的な意味は、任意の個体記号 s, s' に関し、$\bar{s}, \varphi \wedge (y = s), \psi \wedge (y = s') \rangle$（ただし y はフレッシュな（$\varphi, \psi, (\forall x) P(x)$ に登場しない）変数）である。注意として、$P(y)$ を作る際に代入操作が必要である。

[*3] この式は竹内 [7] の定義と論理的な等価である。実際、それのように $\vdash D \rightarrow \exists x Q x$ となる任意の Q について $D \wedge Q$ を考えることと、ここでのように任意の Q について $D \wedge ((\exists x Q x) \rightarrow Q)$ を考えることは等しい。つまり、$D \wedge ((\exists y Q y) \rightarrow Q)$ が成立するならば当然 $\vdash D \rightarrow \exists x Q x$ となる任意の Q について $D \wedge Q$ が証明可能であり、逆に $\vdash D \rightarrow \exists y Q y$ を仮定すると $D \wedge (\exists x Q x \rightarrow Q)$ が導出される。

- 知識様相論理の充足関数の再帰的定義：量化子を持つ文を、代入操作を持つ文に還元することにより、関数を再帰的に定義し、そしてこの定義が帰納的定義になっていることを証明支援系に理解させる（型検査と停止判定にパスする）こと。

2.3 テスト実装例の紹介

本節では、過去の類似の実装例を紹介し、同時に「一見うまくいきそうな例」として、テスト実装例を紹介する

2.3.1 多変数述語の実装の過去の例とその一般化

これまで Agda 上で実装された様相述語論理の例としては、foma.agda[6] がある。これは、Martin-Löf の直観主義型理論にもとづき、様相述語論理（ただし述語は 1 変数のもののみ）のクリプケ意味論を実装している。詳しく言うと、foma.agda では述語を、前述のように、p が変数を持たない命題ならば $p : \mathbf{Set}$ と型付けし、p が 1 変数述語（domain として D をとる）場合は、前述のように、 $p : D \to \mathbf{Set}$ と型付けする。

この実装例を自然に一般化することで、以下のようなテスト実装を得ることができる。すなわち、p' が 2 変数述語の場合は $p' : D \to D \to \mathbf{Set}$、$p$" が 3 変数述語の場合は p" $: D \to D \to D \to \mathbf{Set}$、 と型付けする。

この方針の利点は、以下のとおりである。例として 2 変数述語 $P : \mathbf{Pred}\, 2$ を考えよう。変数をすべて束縛した論理式 $(\forall x)(\forall y)P(x, y)$ において、x に t を代入した場合、論理式は $(\forall y)P(t, y)$ となる。$(\forall x)(\forall y)P(x, y)$ の意味論への解釈は $(x : D) \to (y : D) \to Pxy$ であり、この x に t を代入した集合は $(y : D) \to Pty$ となり、自然な形で通常の述語論理における束縛変数への個体記号の代入を模倣している[*4]。

さて、一般の自然数 n の場合、この方法を一般化し $\mathbf{Pred}\, n$ を再帰的に

- $\mathbf{Pred}\, 0 = \mathbf{Set}$
- $\mathbf{Pred}\, n + 1 = D \to \mathbf{Pred}\, n$

と定義すればよいだろうと考えられる。この場合、$p : \mathbf{Pred}\, n$ のとき、$t : D$ であれば $pt : \mathbf{Pred}\, n - 1$ となり、述語への個体の適用が簡単に定義できる。

2.3.2 述語論理の論理式のデータ型とその解釈の構成

2.3.1 節の方針をすべて採用したと仮定して、話を進めよう。もちろん 2.3.1 節の方針がすべて可能であれば、変数への代入操作等を抽象的にデータ型のコンストラクターとして扱う必要はない。しかし、以下の理由により、本節ではデータ型として述語論理の論理式全体からなるデータ型 **Fml** を定義する。

- より一般的な扱い方を目指す
- 証明のときにパターンマッチなどの支援機能を使用する

今回は簡単のため量化子と変数への代入のみを取り扱うので、**Fml** はコンストラクターとして

- $\mathbf{trans} : \{n : \mathbf{N}\} \to \mathbf{Pred}\, n \to \mathbf{Fml}\, n$
- $\mathbf{subst} : \{n : \mathbf{N}\} \to D \to \mathbf{Fml}\, n + 1 \to \mathbf{Fml}\, n$

[*4] このように、変数 x は実際は D の元 $x : D$ として（Agda 備え付けの変数機構を利用する shallow embedding として）解釈され、また代入操作自体はまったく形式的に扱う必要はない。

- $\mathbf{forall} : \{n : \mathbf{N}\} \to D \to \mathbf{Fml}\, n + 1 \to \mathbf{Fml}\, n$

のみを持つと仮定する。つまり、$\mathbf{Fml}\, n$ は自由変数を n 個もつ述語もしくは論理式である。2.3.1 節での変数と代入の実装が可能であれば、それぞれの解釈は、解釈関数 $\mathbf{int} : \{n : \mathbf{N}\} \to \mathbf{Fml}\, n \to \mathbf{Set}$[*5]に関して、

- $\mathbf{int}\,(\mathbf{subst}\, t\, p) = \mathbf{int}\,(p)t$ （ただし $t : D$）
- $\mathbf{int}\,(\mathbf{forall}\, x\, p) = (x : D) \to (\mathbf{int}\, p)x$

と定義できるはずである。

2.3.3 フレッシュな変数 y の表現について

述語論理では、束縛された変数の箇所を自由変数で置換する作業がよく行われる（これによって全称量化子への代入処理を代行する場合もある）。今回の実装では、それを模倣して、フレッシュな変数を表現する関数 $\mathbf{fr} : \{n : \mathbf{N}\} \to D^n \to \mathbf{Set}$ を postulate によって（天下りに）定義する。意味としては、$\mathbf{fr}(\bar{s})$ は、$\bar{s}$ の任意の構成元とも異なる、新しい変数を表現する D の元であることを表現させたい。したがって同時に、そのことの証明を与える関数

$$\mathbf{frproof} : \{n : \mathbf{N}\} \to (\vec{t} : D^n) \to (i < n) \to (\mathbf{proof}(\bar{t}(i) = \mathbf{fr}(\bar{t}) \to \bot))$$

を定義する。この関数は、長さ n の列 $\bar{t}$ を与えると、$\bar{t} = (t_0, \cdots, t_{n-1})$ の任意の要素 t_i に関して $t_i \neq \mathbf{fr}(\bar{t})$ が成立しているという証明を与える。

2.3.4 充足関数の具体的な定義

その $\mathbf{Fml}$ の上で、前述の様相論理の充足関数 $\models$ は、以下のように定義されると書くことができる。そのデータ型は $\models : \{n : \mathbf{N}\} \to listD \to \mathbf{Fml}\, n \to \mathbf{Fml}\, n \to \mathbf{Fml}\, 0 \to \mathbf{Set}$[*6]であり、定義は再帰的に以下のように行われる。

$$\begin{aligned}
\langle \bar{s}, \varphi, \psi \rangle &\models \mathbf{subst}\, t(\mathbf{trans}\, Q) \\
&= \mathbf{int}\,(\Delta(\bar{s}) \wedge \psi) \to (\mathbf{int}\, Q)t \\
\langle \bar{s}, \varphi, \psi \rangle &\models \mathbf{subst}\, t_0\, \mathbf{subst}\, t_1\, P \\
&= (\langle \bar{s}, \varphi, \psi \rangle \models \mathbf{subst}\, t_1 Q)t_0 \\
\langle \bar{s}, \varphi, \psi \rangle &\models (\mathbf{forall}\, x\, p) \\
&= (\forall ts : D^n) \\
&\quad \to (\forall j)(\forall Q, R : \mathbf{Fml}\, j) \\
&\qquad [\langle \bar{s}, \varphi \wedge ((\exists y)Q(y) \to Q), \psi \wedge ((\exists y)R(y) \to R) \rangle \\
&\qquad\quad \models (\mathbf{subst}\,(\mathbf{fr}(ts))P)ts]
\end{aligned}$$

この定義[*7]は、先に紹介した充足関数の定義をそのまま Agda に実装したものといえる。

注意として、$\models \mathbf{forall}\, x\, p$ の定義は、人間の目には十分簡単なケースへの還元であった。しかし、上の具体的な定義を見る限りでは、コンストラクター $\mathbf{forall}$ を、コンストラクター $\mathbf{subst}$ とフレッシュ変数 $\mathbf{fr}(ts)$

[*5] 実際の実装では、コードの簡略化のため、$\mathbf{int}$ は $\mathbf{int} : \{n : \mathbf{N}\} \to \mathbf{Fml}\, n \to \mathbf{Pred}\, 0$ の型を持つものとして定義したが、$\mathbf{Pred}\, 0$ は実質的に $\mathbf{Set}$ と等しいので、以後 $\mathbf{Set}$ への関数として表記する。

[*6] 簡単のため、透過文脈と不透過文脈はともに同じ型 $\mathbf{Fml}\, n$ を持つと仮定する。

[*7] $\bar{t} = (t_0, \cdots, c_{n-1})$ のとき、$\Delta(\bar{s})$ は $t_0 \neq t_1 \& \cdots \& t_{n-2} \neq t_{n-1}$ と言う式を表す。

による代入操作に還元するため、簡単なケースへの還元とは一見してわからない。この点は、2.4.2 節で検討する。

2.3.5 まとめ

- $\mathbf{Pred}\, n$ は再帰的に定義する（$\mathbf{Pred}\, n + 1 = D \to \mathbf{Pred}\, n$）
- $\models$ を再帰的に定義し、定義式では束縛変数に $\mathbf{fr}(x)$ の値を代入することにより、フレッシュな変数の代入を表現する

2.4 テスト実装例の問題点

2.3.4 節で紹介したテスト実装は、しかし、Agda 上の正しい定義とはなっていない。本節ではテスト例の問題点を分析し、二点を指摘する。

2.4.1 問題点その一：任意有限個の変数を持つ述語のデータ構造として定義することの困難

まず、メタ言語（述語論理）のデータ型の実装に関する困難がある。テスト実装例では、1 変数の場合の述語定義を自然に拡張し、任意有限個の変数を持つ述語のデータ型 $\mathbf{Pred}\, n$ を再帰的に定義しようとしたが、それでは Agda (ver. 2.0.0) の型検査に合格することはできない。

ver. 2.0.0 では、右辺 ($D \to \mathbf{Pred}\, n$) の値域が明示的に（**Set** 等の）ソート （Sort）でないと、型検査に合格しない。この場合、$\mathbf{Pred}\, n$ は再帰的に定義された型であり、ソートではないので、型検査に失敗する。右辺の値域にソート以外のデータ型の代入を許す場合、そのデータ型が再帰法によって循環的に定義されているとすると、一般の再帰法の場合と同様、評価結果が定まらないことが起こりうる。したがって、値域には型判断が停止することが保障されているソートのみが許された。

注意として、型検査アルゴリズムの進歩により、2008 年 3 月現在の最新バージョン ver.2.1.3 では、型検査に合格する。しかし、ver.2.0.0 の検査アルゴリズムに合格しなかったことは、**Pred** の定義の仕方が不必要に複雑で、問題を含んでいることを示唆する。特に、他の証明支援系へ応用する際、この定義がその系で可能であるかどうかが問題となるだろう。

また、通常の論理操作においては、2 変数の述語 $p(x, y)$ について、x と y が自由変数であれば、$(\forall x)(\forall y)p(x, y)$ と $(\forall y)(\forall x)p(x, y)$ がともに導出可能である。$(\forall x)(\forall y)p(x, y)$ においては、最初に変数 x に個体記号を代入し、次に y に代入する。つまり二番目の変数 y の選択は x の選択に依存している。しかし、$(\forall y)(\forall x)p(x, y)$ では順番は逆である。Agda は強い依存型理論に基づいているため、$p : \mathbf{Pred}\, n$ の変数への代入の順番は厳格に決定されており、順番の変更は明示的にはできない。そのため、明示的に両者の意味が異なるようにしたい。

2.4.2 問題点その二：関数の一般再帰的定義の停止判定にまつわる困難

つぎに、対象言語（知識様相論理）の意味論の定義における困難がある。充足関数 $\models$ の 2.3.4 節のような定義では、Agda の停止判定に合格することができない。

2.3.4 節で触れたとおり、$\models$ の定義では $\mathbf{forall}\, x\, Px$ の充足関数による値 $\langle \bar{s}, \varphi, \psi \rangle \models \mathbf{forall}\, x\, Px$ を、それまでの式中（の個体記号の列 $\bar{s}$）に登場しない新しい変数 y（$\mathbf{fr}(\bar{s})$ で表現されている）を変数 x に代入した式 $P(y)$ の値 $(ts : D^n) \to \cdots \models (\mathbf{subst}\, (\mathbf{fr}(\bar{s}))\, P)ts$ に還元している。しかし、証明支援系にとっては、$\mathbf{forall}\, x\, Px$ はコンストラクター **forall** と $x : D$ を含むデータであり、一方 $\mathbf{subst}\, (\mathbf{fr}(\bar{s}))\, P$ はコンストラクター **subst** と $\mathbf{fr}(\bar{s}) : D$ をもつデータであり、データの複雑さの点では似たようなものである。そのため、前

者のケース $\models \mathbf{forall}\, x\, p$ を後者のケース $\cdots \models \mathbf{subst}\,(\mathbf{fr}(\bar{s}))\, P$ に還元する上記の再帰的定義は、Agda の停止判定メカニズムが循環的定義であると判断してしまう。

加えて、量化子と代入操作が入り混じるケース（$\langle \bar{s}, \varphi, \psi \rangle \models \mathbf{forall}\, x\, \mathbf{subst}\, t\, \mathbf{forall}\, z\, Rxtz$ など）では、代入操作の順番を変更する必要があるが、上記関数定義のパターン・マッチでは、この場合を処理することができない。したがって、人間が「意味を考えて」、同じ意味である $\langle \vec{s}, \varphi, \psi \rangle \models \mathbf{forall}\, x\, \mathbf{forall}\, z\, \mathbf{subst}\, t\, Rxtz$ などに書き換えてやる必要がある。

3　多変数述語を持つ述語論理とその意味論の実装

2.4 節でみたように、テスト実装例は一見正しく見えるものの、二つの問題点（任意有限個の変数を型として定義できない、充足関数の一般再帰的定義が停止判定に合格しない）がある。本章では、最初の問題点の解決法として、ベクトルを使った多変数述語の定義を紹介する。

3.1　述語論理の論理式の型の定義

テスト実装例のような、多変数述語の型の再帰的定義が不可能なため、多変数の述語は $X \to \mathbf{Set}$ の形で定義しなければならない。そのため、n 変数述語の解釈のデータ型 $\mathbf{Pred}'$ を、$\mathbf{N} \to \mathbf{Set}1$ の依存型を持ち、

$$\mathbf{Pred}'\, n = D^n \to \mathbf{Set}$$

として、すなわち D の元の長さ n のベクトル D^n を定義域とし $\mathbf{Set}$ を値域とする関数として定義する。

しかし、$\mathbf{Pred}'$ は型理論的に大きな対象（ソートが $\mathbf{Set}1$）であり、他のデータ型（みなソートが $\mathbf{Set}$ である）とソートを合致させるため、$\mathbf{Set} \to \mathbf{Set}1$ の埋め込み関数を定義するなどの煩雑な操作が必要になる。このため、多変数述語を表す述語記号全体のなす型 $\mathbf{PredSym} : \mathbf{Set}$ を、型 $\mathbf{N} \to \mathbf{Set}$ を持つものとして（postulate により）定義する[*8]。イメージとして、$\mathbf{PredSym}$ は構文論上の述語記号の集合であり、$\mathbf{Pred}'$ はその（意味論で）解釈された述語そのものの集まりと考えてほしい。$\mathbf{PredSym}$ のソートは $\mathbf{Set}$ であり、これを使って他の論理式のデータ型 $\mathbf{Fml}, V$ を定義することで、それらのデータ型のソートを全て $\mathbf{Set}$ で揃えることができる。

$\mathbf{PredSym}$ を使用して、先ほどと同じように、述語論理の論理式のデータ型 $\mathbf{Fml}$ を定義する。これはコンストラクター $\mathbf{fml} : \{n : \mathbf{N}\} \to \mathbf{PredSym}\, n \to \mathbf{Fml}\, n$ や $\forall$ などを持つ。

3.2　ベクトルの使用によりもたらされる問題点

$\mathbf{Fml}$ を定義すると同時に、ベクトルを使用して直観主義論理の意味論を定義したい。これは、以下を満たす関数 $\mathbf{int}$ を定義することを意味する。すなわち、$q : \mathbf{Fml}\, n$ の意味論上の値 $\mathbf{int}\,(q)$ は型 $\mathbf{Pred}'\, n$ を持つ、つまり $\mathbf{int}\,(q)$ は長さ n のベクトル D^n から $\mathbf{Set}$ への関数 $\mathbf{int}\,(q) : D^n \to \mathbf{Set}$ として表現されるようなものとしたい。

しかし、量化子を持つ文の解釈に関して、ベクトルを導入したことにより、変数への代入操作は全て一遍に行わないといけなくなり、この点が問題を起こす。つまり、述語論理で複数の量化子を持つ論理式について、量化子のひとつに代入を行う場合、前述のように、$(\forall x)(\forall y)P(x, y)$ に関して、変数 x に $t : D$ を代入した場

[*8] 同時に $\mathbf{PredSym}$ の解釈関数 $\mathbf{val} : \{n : \mathbf{N}\} \to \mathbf{PredSym}\, n \to \mathbf{Pred}'\, n$ を定義する。

合、$(\forall y)P(t,y)$ となる。しかし、述語 P の意味論における解釈 $\mathbf{int}\,(P)$ は $D^2 \to \mathbf{Set}$ の関数であり、$t : D$ を入力することはできない。同時に y への代入項 s も定めて、$\langle s,t\rangle : D^2$ としてやっと $\mathbf{int}\,(p)$ に代入することができる。このような、変数への個別の代入が不可能な体系は、述語論理の実装としては問題があると思われる。

また、2.4.1 節で指摘した、量化の順番の変更も可能にするようにしなければならない。

3.3 ベクトルの操作の導入と意味論の定義

3.2 節で指摘した問題点を解決するため、以下のようなベクトル操作を定義する。

まず、一番問題がない場合として、$p : \mathbf{PredSym}\,n$ にベクトル $\vec{t} : D^n$ を代入する操作を定義し、その意味論上の解釈を定める。ベクトルを代入する操作を $\mathbf{vsubst}\ \vec{t}\,p$ と表記し、解釈は以下のとおりである。

$$\mathbf{int}\,(\mathbf{vsubst}\ \vec{t}\,p) = (\mathbf{int}\,p)\vec{t}$$

次に、量化子を含む文の解釈のため、ベクトルに新しい記号を挿入する操作を定義する。

- $\mathbf{gt}\,(0,\langle a,b\rangle,c) = \langle c,a,b\rangle$,
- $\mathbf{gt}\,(1,\langle a,b\rangle,c) = \langle a,c,b\rangle$,
- $\mathbf{gt}\,(2,\langle a,b\rangle,c) = \langle a,b,c\rangle$,
- $\mathbf{gt}\,(3,\langle a,b\rangle,c) = \langle a,b,c\rangle$.

つまり $\mathbf{gt}\,(k,\vec{t},c)$ はベクトル $\vec{t}$ の k 番目の箇所に、新しい個体 c を付け加えたものに当たる。これは、ベクトルにフレッシュな変数を挿入する際に用いる（フレッシュな変数は、前章の $\mathbf{fr}(x)$ によって表現する）。

ベクトルの操作を使用して、以下のように述語論理の意味論をデータ型として定義しよう。簡単のため、ここでは、$\forall$ の例のみを紹介する。

まず準備として、対象論理において、述語の変数に番号をつける。$p : \mathbf{PredSym}\,n+1$ とすると、p は 0 から n までの番号がついた変数を持つことになる。そして、論理式 $\forall xkp$ は、p の k 番目の変数が束縛されている（そしてその変数は x で表される）と考える。

このとき、$\forall xkp$ は $n+1$ 個の変数のうち一つだけ束縛されているので、n 変数述語である。その意味論における解釈を、以下のように構成する。

$$\mathbf{int}\,(\forall x\,k\,p) = \lambda(ts : D^n) \to (\mathbf{int}\,p)(\mathbf{gt}\,(k,ts,(\mathbf{fr}(ts))))$$

注意として、p はもともと $n+1$ 変数であり、一方 $\mathbf{int}\,(\forall\,x\,k\,p)$ は n 変数である。長さが n のベクトル ts を受け取ると、$\mathbf{gt}$ によってベクトルの k 番目の場所に新しい変数（$\mathbf{fr}(ts)$ で表現される）を挿入し、長さ $n+1$ のベクトルにすることによって $\mathbf{int}\,(p)$ に適用可能にする。また、代入 $\mathbf{fml\text{-}subst}\,x\,k\,p$ についても、全称量化子の場合とまったく同様に、以下のように定義する。

$$\mathbf{int}\,(\mathbf{fml\text{-}subst}\,t\,k\,p) = \lambda(ts : D^n) \to (\mathbf{int}\,p)(\mathbf{gt}\,(k,ts,t))$$

また量化の順番に関しては、ベクトルの並び替え $\mathbf{per}$ を以下のように定義する。

$$\mathbf{int}\,(\mathbf{per}\,p)(\langle t_0,t_1,\cdots\rangle) = \mathbf{int}\,(p)(\langle t_1,t0,\cdots\rangle)$$

3.4 Agda への実装例

最後に、Agda へのデータ型の実装例を紹介する。まず、述語論理のデータ型は以下のように定義される。

```
data Fml : (n : Nat) -> Set where
  fml : {n : Nat} -> Pred n -> Fml n
  forall : {n : Nat} -> Nat -> D  -> Fml (n+1) -> Fml n
  fml-subst : {n : Nat} -> Nat -> D -> Fml (n+1)  -> Fml n
  vsubst : {n : Nat} -> Vec D n -> Fml n -> Fml zero
```

本来は、ここで変数の数が違う述語同士の操作のため、lift など変数の個数を調整する操作が必要になるが、それについては付録を参照されたい。また、意味論への解釈の定義は以下のようになる。

```
int : {n : Nat} -> Fml n -> Pred n
int {n}   (fml p) = \ts -> p ts
int {n}   (forall k p) = \ts -> (int p)(gt k ts (fresh ts))
int {n} (fml-subst t p k) = \xs -> (int p) (gt k xs t)
int {zero} (vsub ts p) = \x -> (int p) ts
```

4 充足関数の再帰的定義の実装例

メタ言語である述語論理に関して、前章では自由変数の個数 n に依存したデータ型 $\mathbf{Fml}\,n$ を定義した。しかし対象言語である様相論理については、同様の定義ではその充足関数を停止判定に合格させることができない。この問題点を解決するため、型に変数についての情報を増やし、述語の変数の個数の他に、束縛されている変数の個数の情報も持った依存型 $V_{n,m}$ を定義する。新たに m という情報が加わったため、m の上の帰納法によって充足関数の定義をしてやれば、Agda にも充足関数の値の計算が有限回で停止することが判定できるようになる。

この章では、まず参考とした λ-計算の実装例を紹介し、そしてそれを応用して $V_{n,m}$ とその上の置換を定義し、それを使用して停止判定に合格するような充足関数を定義する。

4.1 参考例：λ 項のデータ型の Agda 上の定義

Agda 上の束縛変数を持つデータ構造の導入例として、λ-計算の定義がある [2][3]。本節では、その簡略化版を紹介する。まずデータ構造として $\mathbf{Tm}$（ただし $\mathbf{Tm}\,n$ は自由変数を n 個含む λ-項を形式化したデータ型）を定義する。

```
data Tm: Nat -> Set where
  var: {n :Nat} -> Fin n -> Tm n
  lambda : {n : Nat} -> Tm (n+1) -> Tm n
  _*_ : {n : Nat} -> Tm n -> Tm n -> Tm n
```

ここでは、変数の指定は de Bruijn 方式により、$m < n$ となる m を添え字として表記される[*9][*10]。このとき、この上の置換 sbst は以下のように定義される。

```
data sbst: Nat -> Nat -> Set where
  empty : {n:Nat} -> sbst 0 n
  _>_ : {k n :Nat} -> sbst k n -> Tm n -> sbst k+1 n
```

つまり、変数が n 個ある λ-項　$p : \mathbf{Tm}\, n$ に関して、

- empty p は n 個の変数のうち 0 個の変数に代入がされており、
- すでに m 個の変数に代入がされた $\lambda x.p[x] : \mathbf{Sbst}\, m\, n$ に、もうひとつ $t : \mathbf{Tm}\, n$ を代入した結果 $\lambda x.p[x] > t$ は、$m+1$ 個の変数に代入が行われた λ-項となる。

4.2 対象言語のデータ型の定義

4.1 節の λ-計算の実装は、変数の個数だけでなく、代入操作の個数に関する情報も型が含んでいるようなものであった。この λ 計算のデータ型は、停止判定に合格する必要はないため、直接にこの定義が応用できるわけではないが、型情報に変数の個数以外の情報（置換操作の回数）を含めるという点で、非常に示唆的である。それを参考に、本節では対象言語の論理式のデータ型を、変数の個数とその中の束縛変数の個数という二つの情報を含むものとして定義する。

では、多変数述語を持つ論理式のデータ型 $V_{n,m}$ （ただし $m < n$[*11]）を定義しよう。

- V は $V : (n : \mathbf{N}) \to (m < n) \to \mathbf{Set}$ の依存型を持つ。$V_{n,m}$ のうち、 $n : \mathbf{N}$ は述語の持つ変数の数を表し、$m < n$ はそのうち何個の変数が（量化子と代入によって）束縛されているかを表す。
- この型はコンストラクター **trans** を持つ。

$$\mathbf{trans} : \{n : \mathbf{N}\} \to \mathbf{PredSym}\, n \to V_{n,0}$$

  したがって、$\mathbf{trans}\, p$ は束縛された変数が一つもない論理式を表す。
- この上で論理結合子や量化子、また lift などの変数の個数の操作子を導入する。今回は簡単のため、量化子と代入操作のみをコンストラクターとして定義する。
  - $\mathbf{forall} : \{n : \mathbf{N}\} \to \{m < n\} \to D \to V_{n,m} \to V_{n,m+1}$
  - $\mathbf{subst} : \{n : \mathbf{N}\} \to \{m < n\} \to D \to V_{n,m} \to V_{n,m+1}$

例えば、5 変数の述語 p と $t : D$ に関して、$(\forall x)p(x, t, y_0, y_1, y_2)$ という量化と代入を加えた式（自由変数 y_0, y_1, y_2 を持つ）を考えてみよう。このデータ型の表現では、$p : \mathbf{PredSym}\, 5$ に関して、$\mathbf{subst}\, s\, \mathbf{forall}\, t\, (\mathbf{trans}\, p)$ は $V_{5,2}$ という型を持ち、自由変数の個数は 3 個である（テスト実装例では $\mathbf{Fml}\, 3$ という型付けとなる）。

[*9] 本来は関数 $\mathbf{var}\,(x)$ の引数は、依存型 Fin n (Finite sets) の元が使われる。Fin n は「n より小さい自然数の集合」（つまり $\{0, 1, \cdots, n-1\}$）と同一視される。これを使用するのは、$\mathbf{var}\,(x)$ の x が n より明示的に小さな自然数であることを表すためである。しかしこの論文では、簡単のため Fin n は使用せず、自然数 m（ただし $m < n$）で代用する。

[*10] $\lambda x.x$ は $\lambda(\mathbf{var}\, 0)$ と表記され、$\lambda y \lambda x.x * y$ は $\lambda(\lambda(\mathbf{var}\,(1)) * \mathbf{var}\,(0))$ と表記される。注意として、今回のわれわれの実装は、Agda の変数機構を使用する shallow embedding のため、直接 de Bruijn 方式の変数の使用はしなかった。

[*11] 4.1 節と同じく、m は n に依存し、実際は Fin n を使用するが、今回は簡略化のため自然数を使用した。

この新たに付加された情報によって、停止判定に合格するような充足関数の定義が可能となる。それを次節で紹介する。

4.3 充足関数の定義

この節では、対象言語である知識様相論理の充足関数（意味論）の定義を、メタ言語である述語論理上の再帰法により行う。この再帰法は、4.2 節で定義した束縛変数の個数の情報を含む型 $V_{n,m}$ の m についての帰納的定義となるので、有限回で再帰が停止することが Agda にも判定できる。このことは、$V_{n,m}$ という型に情報（束縛変数の個数）が付加された事により可能となった。以下、再帰的定義の詳細と、充足関数の計算の実例を紹介し、最後にその定義が Agda の停止判定を合格することを解説する。

4.3.1 充足関数の再帰的定義

充足関数の定義は、任意の論理式について、m 上の帰納法によって束縛変数を排除していき、$m = 0$ になると **Set** の値を出力する。したがって、この式変形操作を有限回繰り返せば計算が停止することが、明示的に判断できる。

充足関数 **Sat** の型は

$$\mathbf{Sat} : \{k : \mathbf{N}\} \to listD \to \mathbf{Fml}\, k \to \mathbf{Fml}\, k \to (n : \mathbf{N}) \to (m < n) \to V_{n,m} \to listD \to \mathbf{Set}$$

である[*12]。詳細については以下を参考にされたい。

- 最初の $listD$ は、D の元によるリストのデータ型で、個体についての共有概念を表す。二つの $\mathbf{Fml}\, k$ はそれぞれ透過文脈と非透過文脈を表している。
- m と $V_{n,m}$ は依存型であり、それぞれ n と n, m の選択に依存する。
- 最後の $listD$ は、変数へ代入される D の元の、再帰法で還元される際に除去された元からなるリスト型である[*13]
- つまり $\mathbf{Sat}\,(\vec{t}, \varphi, \psi, n, n, P, ())$ は、2.3 節の表記法では $\langle \vec{t}, \varphi, \psi \rangle \models P$ を表している。

ここで、表記法について注意する。以下、リストの延長を :: で表す（つまりリスト $(s_{n-1}, \cdots, s_0)$ の先頭に s_n を付け加えたリストを $(s_n) :: (s_{n-1}, \cdots, s_0)$ で表現する）。

では、**Sat** による値の計算の仕方を見てみよう。$(\mathbf{trans}\ p) : V_{n,0}$ に関して、 それを量化子と代入を用いて閉論理式にした式の充足値を、**Sat** で計算してみる。最初のステップにおいて、リスト $\bar{s}$ は空リストである。そして、テスト実装の際に問題となった全称量化子の除去は、以下のように定義される。

$$\begin{aligned}\mathbf{Sat}\,(\vec{t}, \varphi, \psi, n, m+1, (\mathbf{forall}\ x\, p), \bar{s}) = {} & (Q, R : \mathbf{Fml}\, 0) \\ & \to \mathbf{Sat}\,(\vec{t}, \varphi \wedge ((\exists x) Q(x) \Rightarrow Q), (\psi \wedge ((\exists x) R(x) \Rightarrow R), \\ & \qquad n, m, p, (\mathbf{fr}(\vec{t}, \bar{s})) :: \bar{s})\end{aligned}$$

また、変数への代入操作に関して、その再帰法による還元は以下のように定義される。まず、$p : V_{n,m}$ と仮定

[*12] 値域の型は本来は $(j : \mathbf{N}) \to \mathbf{Pred}\,(k + j)$ であるが、再帰的定義に関係がないため、ここでは簡単のため **Set** とした。

[*13] 実際には長さ n のベクトル（ダミー元含む）で実装されている。後述のように、$m = 0$ になった際、代入はベクトルのみ可能である。しかし、本論文では簡単のためリスト型とする。

する。このとき

$$\begin{aligned}&\mathbf{Sat}\,(\vec{t},\varphi,\psi,n,m+1,(\mathbf{subst}\,t\,p),\bar{s})\\&=\mathbf{Sat}\,(\vec{t},\varphi,\psi,n,m,p,(t)::\bar{s})\end{aligned}$$

上の **subst** と **forall** の除去を繰り返して $m=0$ になった時点で、対象言語の論理式は、もう束縛変数を持たない、自由変数のみとなっているはずである。また、その際には (b) のリストも長さが n となっているはずである。従って、リストをベクトル（以下では $\vec{t}$ と表記する）に変換し、

$$\mathbf{Sat}\,((\vec{t},\varphi,\psi,n,0,(\mathbf{trans}\,p),\vec{t})=\mathbf{int}\,(\mathbf{vsubst}\,\vec{t}\,p)=(\mathbf{int}\,(p))\vec{t}$$

つまり、**Sat** は最終的に、対象言語の論理式を、**int** を使用して、メタ論理の意味論の対象 $(\mathbf{int}\,(p))\vec{t}:\mathbf{Set}$ に解釈する。代入操作の順番や複数量化の順番に関しては、前章で定義したベクトルへの操作を使用することで制御する。

4.3.2 充足関数の計算の実例

最後に、充足関数の計算の具体的な例をみてみよう。

まず、論理式の型 $V_{n,m}$ における論理式の構成法を、通常の述語論理におけるやりかたと比較する。通常の述語論理で、2 変数述語 Q と $t:D$ から、$(\forall x)Q(x,t)$ という閉論理式を構成する操作と同等のことを V の式に関して行おう。

- $q:\mathbf{PredSym}\,2$ と仮定する。このとき、$(\mathbf{trans}\,q):V_{2,0}$ となる。これが自由変数を 2 つ持つ述語 $Q(x,y)$ を表す記号である。
- 一つ目の変数 x を **forall** で量化した式は $\mathbf{forall}\,x\,(\mathbf{trans}\,q)$ となり、この式の型は $V_{2,1}$ となる。これに対応する述語論理での式は $\forall xQ(x,y)$ である。
- この式に唯一残った自由変数に、$t:D$ を代入してみよう。我々の表記では $\mathbf{subst}\,t\,\mathbf{forall}\,x\,(\mathbf{trans}\,q)$ となり、これは $V_{2,2}$ の型を持つ。この式に対応する述語論理の式は閉論理式 $\forall xQ(x,t)$ である。

さて、この論理式 $\mathbf{subst}\,t\,\mathbf{forall}\,x\,(\mathbf{trans}\,q)$ の充足関数 **Sat** による値は、以下のように計算される。

$$\begin{aligned}\mathbf{Sat}\,((\vec{t},\varphi,\psi,2,2,(\mathbf{subst}\,t\,\mathbf{forall}\,x\,(\mathbf{trans}\,q)),())&=\mathbf{Sat}\,((\vec{t},\varphi,\psi,2,1,(\mathbf{forall}\,x\,(\mathbf{trans}\,q)),(t))\\&=\mathbf{Sat}\,((\vec{t},\varphi,\psi,2,0,(\mathbf{trans}\,q),(\mathbf{fr}(\vec{t}),t))\\&=(\mathbf{int}\,(q))(\mathbf{fr}(\vec{t}),t)\end{aligned}$$

かくして、有限ステップの変形で $(\mathbf{int}\,(q))(\mathbf{fr}(\vec{t}),t):\mathbf{Set}$ が値として出力される。

4.3.3 停止判定に合格する理由

4.3.1 節の定義において、ポイントとなるのは以下の二箇所である。

(a) **forall** と **subst** はどちらも $V_{n,m+1}\to V_{n,m}$ の型を持ち、添え字が $m+1$ から m へと明示的に 1 減っている。

(b) 個体のリスト $(t_0,\cdots,t_{n-1})$ を構成する： 再帰法の任意のステップ $k<n$ において、

(i) 全称量化子の場合は $t_k=\mathbf{fr}((t_0,\cdots,t_{k-1}))$ として、

(ii) **subst** の場合は代入される元 $t:D$ により $t_k=t$ とする。

$m = 0$ になったときに、このリストをベクトル D^n に変換し、$\mathbf{int}\,(p) : D^n \to \mathbf{Set}$ に入力することで、**Set** の型を持つ出力を得る。

(a) は、再帰法による計算が、有限回で必ず停止することを明示的に保証する。このことは Agda も理解することができ、Agda の停止判定に合格することができた。また、**forall** の除去と **subst** の除去は、両者ともまったく同じように扱うことができるので、複数の量化子と置換を持つ論理式に関しても、問題なく充足関数の計算を進めることができる。(b) は、2.4.2 節のような事態が起こらず、量化子と置換の順番にかかわらずに計算が進められることを保障する。

4.4 Agda への実装例

最後に、参考のために、Agda によるコードを紹介する。データ型の定義は以下の通り。

```
data V : (n : Nat)(fi : Fin (s n)) -> Set where
  trans  : {n : Nat} -> Fml n -> V n fzero
  Forall : {n : Nat} -> {fj : Fin (s n)} -> D -> V (s n) (fin-liftup fj)
                                                      -> V (s n) (fs fj)
  subst  : {n : Nat} -> {fi : Fin (s n)} -> D -> V (s n) (fin-liftup fi)
                                                      -> V (s n) (fs fi)
  perm   : {n : Nat} -> {fj : Fin (s n)}  -> V (s (s n)) (fs (fs fj))
                                                  -> V (s (s n)) (fs (fs fj))
```

注で述べたように、実際の実装では $m < n$ としてデータ型の Fin n を使用している[*14]。関数定義は、長いため省略する。

5 まとめ

本論文では、Martin-Löf の直観主義的型理論に基づいた厳格な型付けを持つ証明支援系 Agda 上で、知識様相論理の意味論（充足関数）を定義した例を紹介した。目的の充足関数は、述語論理の証明可能性関係によって（原始再帰的にではないが）再帰的に定義されており、この関数の実装には、任意有限個の変数を持つ述語を持つ述語論理と、一般の再帰的定義を可能とする様相論理の論理式のデータ型の定義が不可欠であった。

任意有限個の変数を持つ述語を含みさらに量化子による変数の束縛が可能となるデータ構造としての述語論理の定義は困難であり、よく知られたやり方では Agda の型検査に合格しない。さらに、束縛された変数への具体的な値を代入する操作を証明支援系が解釈することにも困難が伴い、量化子を含む論理式によって関数を再帰的に定義しようとして、たとえそれが人間の目には十全な定義であっても、証明支援系はその関数の停止判定に失敗することが起こった。

本論文においては、多変数述語はベクトルを使用し定義した。また、各種ベクトルの操作を定義することで、述語論理の意味論を定義することができた。そして、再帰的定義を可能にする様相論理の論理式の型の定義には依存型理論 (dependent type theory) を使用した。すなわち、自由変数の個数と、そのうちの束縛された変数の個数を、同時に型に情報として含むように、多変数述語を含む論理式のデータ型を定義した。また、

[*14] fin-liftup は、fj: Fin n を、そのまま Fin $n+1$ に持ち上げるために使う関数である

束縛された変数の個数の上での帰納法により、知識様相述語論理の充足関係を関数として定義した。このことにより、関数定義が明示的に有限回で停止することを証明支援系に理解させることができ、関数の停止判定に成功した。

束縛された変数を持つデータ構造において、再帰的に関数を定義する際の困難は、Agda だけでなく、すべての型付けを持つ言語に共通している。その点で、今回の実装例は、他の証明支援系においても実装の際の参考になるものと期待される。

参考文献

[1] Conor McBride, Thorsten Altenkirch and James McKinna. Why Dependent Types Matter. *Submitted to ICFP 2005.*

[2] Nils Anders Danielsson. A Formalisation of the Correctness Result From "Lightweight Semiformal Time Complexity Analysis for Purely Functional Data Structures". *Technical report 2007:16*, Department of Computer Science and Engineering, Chalmers University of Technology.

[3] Conor McBride. Type-Preserving Renaming and Substitution. Preprint.

[4] Bengt Nordström, Kent Petersson, Jan M. Smith. Programming in Martin-Löf's type theory. Oxford University Press. 1990.

[5] Ulf Norell. Towards a practical programming language based on dependent type theory. Ph.D. thesis. Chalmers University of Technology. 2007.

[6] 岡本圭史. foma.agda.
http://unit.aist.go.jp/cvs/Agda/foml.agda

[7] 竹内泉. 伝聞の様相論理. 科学哲学 vol.39-2 (2006): pp.57-69.

付録: lift の定義

本文中では取り上げなかったが、メタ言語となる述語論理は、$\&, \vee$ などの、二つの命題を一つにまとめる論理結合子を持つ。例えば $\&$ は

$$\& : \{n : \mathbf{N}\} \to \mathbf{Fml}\, n \to \mathbf{Fml}\, n \to \mathbf{Fml}\, n$$

という型をしていると考える。さて、このとき問題となるのは、$\&$ を変数の数の違う二つ述語に適用したいときである。例えば $P : \mathbf{Fml}\, 1$ かつ $Q : \mathbf{Fml}\, 2$ のとき $P\&Q$ を何変数と考えればよいのだろうか。

このとき、以下のオペレーター lift を使用する。

$$\mathbf{lift} : \{n : \mathbf{N}\} \to \mathbf{Fml}\, n \to \mathbf{Fml}\, n + 1$$

上の場合、まず P に **lift** を適用して 2 変数述語とする（ダミーの変数を一つ増やすと考えてよい）。

$$\mathbf{lift}P : \mathbf{Fml}\, 2$$

それから $\mathbf{lift}P$ と Q を $\&$ で連結すればよい。同様に、$\forall x(Px\&Qxy)$ を作りたいときは **lift** により P を 2 変数にしてから $\forall$ で x を束縛する。また、$\forall x(Px\&Qyz)$ を作りたいときは **lift** により P, Q をともに 3 変数にしてから $\forall$ で x を束縛すればよい。

さて、**lift** の意味論での解釈は以下のように定義する。

```
int (fml-lift p)   = \xs -> f xs p
  where
f : {n : Nat} -> Vec D (s n) -> Fml n -> Set
f (x ++ ts) p = (int p) ts
```

すなわち、

$$\mathbf{int}\,(\mathbf{lift}\,p)(t_0, t_1, \cdots) = \mathbf{int}\,(p)(t_1, \cdots)$$

と、先頭の変数をダミーとして排除するやりかたで解釈する。

再帰的定義を可能にする述語論理の証明支援系上の実装 (Preliminary Version)
(算譜科学研究速報)
発行日 2008 年 3 月 13 日
編集・発行：独立行政法人産業技術総合研究所システム検証研究センター
同連絡先：〒563-8577 大阪府池田市緑丘 1-8-31
e-mail：informatics-inquiry@m.aist.go.jp

An implementation of predicate logic in a proof assistant
for supporting recursive definitions (Preliminary Version) (in Japanese)
(Programming Science Technical Report)
March 13, 2008
Research Center for Verification and Semantics (CVS)
Ikeda Site
National Institute of Advanced Industrial Science and Technology (AIST)
1-8-31 Midorigaoka, Ikeda, Osaka, 563-8577, Japan
e-mail: informatics-inquiry@m.aist.go.jp

AIST01-J00022-80