ブール意味論と選択関数

石田　基広

2003 年 9 月

1　はじめに

1.1　本稿の趣旨

本稿は, ブール代数に基づく自然言語の意味論に関する覚え書きである. ここでは特に Yoad Winter [15] の提案の中から,「選択関数」の適用に関する問題を取りあげる.

自然言語の形式的研究において, しばしば議論となる問題に量化子のスコープがある. ところで形式意味論には, これを「選択関数」によって解決しようとする立場がある. この方策では, 選択関数が「自由変数」として導入されるが, Winter 自身が指摘するように, この自由変数が「モデル理論的意味論」においてどのような「対象」を指示するのか, という問題があらたに生じる.

ここでは自由変数としての選択関数と, モデル理論的意味論の整合性について, Winter に基づきつつ考察をすすめる. すなわち本稿は, 最近のブール意味論の展開についての解説を意図している. したがって技術的な面については可能な限り補足を試みたつもりであるが, 紙幅の都合上, 初等程度の数理論理学, ブール代数, 形式意味論を前提として話を進めざるえない. これらについては末尾に参考文献をかかげておく. [1, 2, 3, 4, 5, 7, 6, 10, 11, 12, 13]

2 選択関数

2.1 自然言語におけるスコープ

はじめに選択関数, またそれが必要とされる経験的事実について触れる. Winter から次の英文をあげる. [15, pp. 85–89]

(1) If some building in Washington is atacked by terrorists
then US security will be threatened.

この英文の通常の解釈は, 「ワシントンに (特別な) ビルがあって (つまりホワイトハウスだが), これがテロリストに攻撃されたならば, アメリカの安全保障が脅かされる」というものであろう.

いまこの英文を「通常の」述語論理式によって記述しようとすれば, 次のような式が可能であろう.

(2) $[\exists x[\mathbf{building}'(x) \wedge \mathbf{attacked}'(x)]] \rightarrow \mathbf{threatened}'(\mathbf{security}')$

ここでブラケット ([ ]) は変数 x が束縛されるスコープを表している. しかしながらこの式は, (1) の英文の「直観的」に正しい解釈を表していない. すなわち, (2) の式の意味するところは, 「(なんであれ) テロリストに攻撃されるようなビルがあれば, アメリカの安全保障が脅かされる」というものであり, 下に示す (われわれの「常識」に基づく) 解釈とは一致しない.

(3) $\exists x[\mathbf{building}'(x) \wedge \mathbf{attacked}'(x) \rightarrow \mathbf{threatened}'(\mathbf{security}')]]$

問題はこの式で変数 x の束縛されるスコープが条件文を越え出ていることである. あるいは言い方を変えれば, 文全体が存在量化子のスコープの中におさまってしまっている. すなわち「島」の条件違反となっているのである (「島」の条件, またその経験的根拠については本稿では触れない). このように indefinite が「島」を越えた解釈をともなうことは自然言語に一般的な現象と考えられる. しかし「島」もまた, 自然言語のスコープを説明する重要な概念であり, (3) の解釈との整合性が問われてきたのである.

この問題は, そもそも存在量化の機構に関する問題にも発展しうるのだが [*1],

[*1] 補足すれば Winter は simple definite は determiner ではなく, modifier であるとの議論を展開している. さらに Reinhart とは異なり, definite の Generalized Quantifier としての解釈をあらため, 「選択関数」に統一することを提案している.

Winter は Reinhart [9] に基づき,「選択関数」を用いたスコープ問題の解決を提起している.

2.2 選択関数の定義

今 Reinhart にそった「選択関数」の定義を挙げる. [15, p.89]

> Let E be a non-empty set. A funciton $f : \wp(E) \to E$ is a *choice function* iff for every $A \subseteq E$: if A is not empty then $f(A) \in A$.

これは集合論に見られる一般的な選択関数を表したものにすぎない. [13, p. 169] すなわち選択関数 f は, 空ではない集合から, それが含む要素を選び出す. また選択関数はタイプ理論では以下のように定義される. ここで τ は任意のタイプを表す. つまり以下の定義は, オブジェクト指向流に言えば「多態」である.

(4) $CH_{((\tau t)\tau)t} \stackrel{\text{def}}{=} \lambda f_{(\tau t)\tau}.\forall P_{\tau t} \neq \emptyset\,[P(f(P))]$

この関数定義により, 先の英文のふたつの解釈は次のように表される.

$[\exists f[CH(f) \wedge \mathbf{attacked}'(f(\mathbf{building}'))]] \to \mathbf{threatened}'(\mathbf{security}')$

(5) $\exists f[CH(f) \wedge [\mathbf{attacked}'(f(\mathbf{building}')) \to \mathbf{threatened}'(\mathbf{security}')]]$

重要なのは選択関数を導入した解釈 (5) が「島」を違犯していないことである. つまり definite は元の位置のまま解釈されているのである. このように選択関数は, 自然言語の「島」と, これに違犯した解釈を迫る indefinite を両立させる方策として注目されるのである.

ただし選択関数の定義にはふたつの問題点が指摘される. 第 1 に, この例文で building を *et* のタイプとした場合, f(*building*) の指示するタイプは e となるが, このタイプは boolean ではない. 第 2 に, ここで使われるシンボル f や $\exists f$ とモデル理論的対象との対応である. ここでモデル理論的対象とは何かについて白井 [12, p.11] から引用する.

> 言語表現は, 一般的に,「世界」のなんらかの構成体に対応づけられることになる. この対応づけを組織だった仕方で行なうためには,「世界」を形式的に構成する明確な記述の枠組がなければならない. こうした要求とうまく合致するのが, モデル理論的意味論で発達してきた方策である. そこでは, 通常, 集合論の装置を用いて,「世界」に対する数学的なモ

デル (model) が構築される. そして, 言語表現は意味解釈されて, 最終的には, このような**モデル理論的対象** (model-theoretic object) に対応づけられることになる.

本稿ではこのうち第 2 の問題, すなわち自由変数を擬似的 (mimic) に回避するシステムについての Winter の戦略を以下に概略する.[*2]

3 自由変数の回避

3.1 COND

自由変数を含む表現は関数によって擬似的に実現できる. たとえば some woman は $(et)t$ タイプの表現であるが

$$<f_{(et)e}>(\mathbf{woman}')$$

この「表現」は, 自由変数を含み, モデル意味論との整合性が問題となる. ところが, これは λ 関数を導入することによって次の関数を直接「指示」することができる.[*3]

(6) $\lambda f_{(et)e}.<f>(\mathbf{woman}')$

したがってこの関数は, たとえば述語 smile と合成されるとき (つまり some woman smiles は), $((et)e)t$ のタイプを導出する.

(7) $\lambda f_{(et)e}.(<f>(\mathbf{woman}')(\mathbf{smile}'))$

すなわちこれは次の演繹である.

$$((et)e)((et)t),\quad et \vdash ((et)e)t$$

あえて補足すれば, これは $((et)e)((et)t)$ のタイプと et のタイプを合成し, $((et)e)t$ のタイプが派生することを表している. 直観的に説明すれば, ここで $f_{((et)e)((et)t)}$ は関数であるが, 先に第 2 引数である $((et)t)$ と述語の et が合成され, その結果である t と選択関数とで $(et)e)t$ のタイプを派生する. この

[*2] 第 1 の問題, すなわち選択関数を boolean タイプの枠組みに適合させうることについての証明は [15, p.109] を参照.

[*3] ここで < > は $((et)t)((et)t((et)t))$ タイプの演算子である. すなわち選択関数から, 通常の determiner を導出する関数として定義されている. [15, p.111]

ような推論規則は,「自然演繹」における「仮定」とその「キャンセル」によって表すことができる. Winter は van Benthem [14] にならって, これを COND(*Conditionalizaiton*) と呼んでいる.

(8) $\dfrac{\begin{array}{c}[t]^i\\ \delta\end{array}}{\tau\delta}\ COND^1$ (仮定 i をキャンセル) $\qquad \dfrac{\begin{array}{c}[x]^i\\ y\end{array}}{\lambda x.y}\ COND^1$

ここで仮定が「キャンセルされる」ことにより, 仮定が式の外へと「浸透 percolate」する. そこで COND を用いて, 先の英文 (7) は次のように表される.

$$\dfrac{\dfrac{\dfrac{((et)e)\ ((et)t)\quad [((et)e)]^1}{(et)t}\ APP \qquad et}{t}\ APP}{\dfrac{t}{((et)e)t}\ COND^1} \qquad \dfrac{\dfrac{\dfrac{\lambda f.<f>(\mathbf{woman}')\quad [g]^1}{<g>(\mathbf{woman}')}\ \mathbf{smile}'}{<g>(\mathbf{woman}')\ (\mathbf{smile}')}}{\lambda g.<g>(\mathbf{woman}')\ (\mathbf{smile}')}$$

この派生後も, 選択関数は式の外へと浸透している. すなわち (6) の式の選択関数 f は smile との合成後の式 (7) においても, 後で定義する束縛サイトに「上昇している」のである. Winter の主張の重要な点は, この派生において $[(et)e]$ はメタレベルの変数にすぎず, したがって意味「表示」の一部とはならないということにある.

3.2 TYPE, ACOND

ただし COND の条件は弱すぎるため, たとえば他動詞の (to) see every woman を展開するとき, 次のような派生を可能にしてしまう.

$$\dfrac{\dfrac{\dfrac{e\ (et)\quad [e]^1}{et}\ APP \qquad (et)t}{t}\ APP}{\dfrac{t}{et}\ COND^1} \qquad \dfrac{\dfrac{\dfrac{\mathrm{see}'\quad [y]^1}{\mathrm{see}'(y)}\ APP \qquad \mathrm{every}'(\mathrm{woman}')}{\mathbf{every}'(\mathbf{woman}')(\mathbf{see}'(y))}\ APP}{\lambda y.\mathrm{every}'(\mathrm{woman}')(\mathrm{see}'(y))}\ COND^1$$

この式の表すところは (to be) seen by every woman である. そこで COND を制限する必要があるが, Winter は次の方策をとっている. まず以下の自明な公理が定義される. ここで τ はタイプを表す. また x は τ の指示対象である.

(9) (*AXIOM*) $\quad \tau \vdash \tau \qquad$ 意味論: $x \Rightarrow x$

次に「適用」Application に関する Ajdukiewicz 計算の規則がシークエント計算式として与えられる. [8, p. 550]

$$(10)\quad (R_1)\ \frac{\Gamma \vdash \tau\sigma \quad \Delta \vdash \tau}{\Gamma, \Delta \vdash \sigma}\ APP \qquad \text{意味論:}\ \frac{X \Rightarrow x \quad Y \Rightarrow y}{X, Y \Rightarrow x(y)}$$

すなわち (空ではない) タイプの列 Γ が $\tau\sigma$ を導き, また同じく (空ではない) タイプの列である Δ が τ を導くのであれば, このとき, Γ と Δ というふたつのタイプの列から, τ を導いて構わない.

同じく「置換」permutation の規則はシークエント計算で次のようになる.

$$(11)\quad \frac{\Gamma \vdash \tau}{\Pi(\Gamma) \vdash \tau}\ PERM \qquad \text{意味論:}\ \frac{X \Rightarrow x}{\Pi(X) \Rightarrow x}$$

すなわち Γ 列の中でタイプの順番は自由に入れ換えて構わない.

次に「浸透」を可能にするため, まずタイプ定義が修正される. すなわち従来のタイプに加えて, あらたに $\rightarrow$ 付きの関数であるタイプが定義される.

$$(12)\quad \begin{array}{l} 1.\ e \in \textbf{TYPE} \text{ and } t \in \textbf{TYPE} \\ 2.\ \tau \in \textbf{TYPE} \text{ and } \sigma \in \textbf{TYPE} \text{ and } (\tau \rightarrow \sigma) \in \textbf{TYPE} \end{array}$$

ここで $D_{\tau\sigma}$ と $D_{\tau\rightarrow\sigma}$, さらに $D_\sigma^{D_\tau}$ の三つの指示対象は同一である. 違いはそれらの「構成原理」にある. すなわち $\rightarrow$ 付きタイプは, 項を「浸透」させる効果をもたらす.

たとえば $((et)e) \rightarrow ((et)t)$ では第 1 引数の選択関数を仮定として保留し, 第 2 引数の $(et)t$ と et から, t が派生可能である. その上で, この t に, 保留された選択関数は COND によってあらためて導入されるのであるが, そのとき変数を束縛サイトに位置させる必要がある. これを可能にするため, COND に制限を加えたあらたな規則, ACOND(*Argument Conditionalization*) が導入される.

$$(13)\quad (R_3)\ \frac{\Gamma,\ \sigma_1 \vdash \sigma_2}{\Gamma,\ \tau \rightarrow \sigma_2 \vdash \tau \rightarrow \sigma_2}\ ACOND \qquad \text{意味論:}\ \frac{X,\ y_1(x_\tau) \Rightarrow y_2(x_\tau)}{X, y_1 \Rightarrow y_2}$$

ただし, ここで τ タイプの指示対象 x は, 通常の述語計算の場合と同様, 変数条件を充たしていなければならない.

また ACOND では, $\rightarrow$ によるタイプ形成は自由に行なわれない. たとえば COND 規則では, 以下に示すように $e \vdash (e \rightarrow t) \rightarrow t$ が導出可能だが (以下の派生は自然演繹による),

$$\cfrac{\cfrac{e,\quad [e \rightarrow t]^1}{t}\ APP}{(e \rightarrow t) \rightarrow t}\ COND^1$$

ACOND 規則の場合, 同じ派生を試みようとすると,

$$\frac{e,\ \ (e \to t) \vdash t}{e,\ \ e \to (e \to t) \vdash (e \to t)}$$

となるか

$$\frac{e,\ (e \to t) \vdash t}{e,\ (e \to t) \to (e \to t) \vdash (e \to t) \to t}$$

となってしまう.

また → タイプの関数は「選択関数」を含む派生にのみ適用される. すなわち「辞書」項目には適用されない.

これを先の英文 (7) にあてはめると

$$\frac{et,\ \ (et)\,t \ \vdash\ t}{et,\ \ ((et)\,e) \to ((et)\,t) \ \vdash\ ((et)\,e) \to t}\ ACOND$$

$$\frac{\mathbf{smile}',\ \ (\lambda f.\, <f> (\mathbf{woman}'))(g) \Rightarrow (\lambda f.\, <f> (\mathbf{woman}')(\mathbf{smile}'))(g)}{\mathbf{smile}',\ \ \lambda f.\, <f> (\mathbf{woman}') \Rightarrow \lambda f.\, <f> (\mathbf{woman}')(\mathbf{smile}')}$$

という結果が得られる. 簡単に説明すれば, $(et)\,t$ を et に適用した結果, t が得られるのであれば, $(et)\,t$ を出力する関数 $((et)\,e) \to ((et)\,t)$ を et に適用した結果として, $((et)\,e) \to t$ を導出して構わないというのがこの規則である.

3.3 SAT

次に $(et)\,t$ のタイプの量化子と $e\,(et)$ のタイプの他動詞の構成を可能にするための操作 *Saturation* を定義する. たとえば $(\tau\,t)\,t$ タイプの量化子と, 「関係」を表し, 最初の項が τ であるタイプとが結合する. この操作は次のように (再帰的に) 定義される.

(14) **Arity of boolean types**

Type t is of arity 0. Let σ be a boolean type of arity n. Then any boolean type $\tau\delta$ or $\tau \to \sigma$ is of arity n.

ここで $\tau \to \sigma$ は, 対応する $\tau\sigma$ が boolean である限りにおいて, 同じく boolean であると見なされる.

次に, この操作の意味論が定義される. これは関数 SAT によって定義される. SAT は量化子と n 項関係を引数としてとり, $n-1$ 項関係を導出する.

(15) **Saturation**

For $Q \in D_{(\tau t)t}$ and $R \in D_{\tau\sigma}$, where σ is a boolean type of arity $n \geq 0$:
$SAT(Q,R) \stackrel{\text{def}}{=} \lambda x_1 \ldots \lambda x_n.Q(\lambda x.R(x)(x_1)\ldots(x_n))$

また Saturation に関する推論規則は以下のようにまとめられる.

(16) (R_4) $\dfrac{\Gamma \vdash (\tau t)t \quad \Delta \vdash \tau\ op\ \sigma}{\Gamma,\Delta \vdash \sigma}\ SAT \qquad \dfrac{X \Rightarrow Q \quad Y \Rightarrow R}{X,Y \Rightarrow SAT(Q,R)}$

ここで op は空か, あるいは $\rightarrow$ を表し, また σ は項の数が $n \geq 0$ である.

たとえば関数 SAT に (単数の) 量化子と他動詞を適用するならば以下のようになる.

For $Q_{(et)t}$, $R_{e(et)}$,
$$SAT(Q,R) = \lambda x_1.Q(\lambda y.R(y)(x_1))$$

3.4 ECC

最後に, 選択関数を「束縛」するための演算子が定義される.

(17) **Existential Choice Closure**

$$ECC_{(((et)e)t)t} \stackrel{\text{def}}{=} \lambda A_{((et)e)t}.A \cap CH \neq \emptyset$$

4 実例

では, ここからは Winter に挙げられている派生をより細かく見ていく.

(18) If some woman admires some man then everything's OK

ここで構成要素の辞書は次のように与えられる.

if	$t(tt)$	$\mathbf{if}' = \lambda\varphi_t.\lambda\psi_t.\varphi \rightarrow \psi$
some	$((et)e) \rightarrow ((et)((et)t))$	$\mathbf{some}' = \lambda f_{(et)e}.\lambda A_{et}.\lambda B_{et}.\ <f>(A)(B)$
woman	et	$\mathbf{woman}'$
admires	$e(et)$	$\mathbf{admire}'$
man	et	$\mathbf{man}'$
then	tt	$\mathbf{then}' = \lambda\varphi_t.\varphi$
everything's OK	t	$\mathbf{ok}'$

ここで some は選択関数から determiner への関数である. そしてこの関数のタイプには → が用いられており, これにより選択関数は束縛サイトまで「浸透」することが可能となっている.

$$
\begin{aligned}
SAT(ECC, \mathbf{some}') &= \lambda A_{et}.\lambda B_{et}.ECC(\lambda g_{(et)e}.\mathbf{some}'(g)(A)(B)) \\
&= \lambda A.\lambda B.(\lambda X_{((et)e)t}.X \cap CH \neq \emptyset)(\lambda g.\mathbf{some}'(g)(A)(B)) \\
&= \lambda A.\lambda B.\exists f[CH(f) \wedge \mathbf{some}'(f)(A)(B)] \\
&= \lambda A.\lambda B.\exists f[CH(f) \wedge <f>(A)(B)] \\
&= \lambda A.\lambda B.A \cap B \neq \emptyset \\
&= \mathbf{E}
\end{aligned}
\tag{19}
$$

補足をすれば, 上の派生図の 2 行目で ECC が $\lambda A.\lambda B.(\lambda X_{((et)e)t}.X \cap CH \neq \emptyset)$ に置き換えられ, $\lambda X.X$ に $\lambda g.\mathbf{some}'(g)(A)(B)$ が代入されているわけである. つまり

$$\lambda A.\lambda B.(\lambda g.\mathbf{some}'(g)(A)(B) \cap CH \neq \emptyset)$$

という式が構成される.

これはすなわち選択関数全体の集合と, $\lambda g.\mathbf{some}'(g)(A)(B)$ との積が空集合ではないということであるから, かならずある選択関数 g が存在する. それを表したのが $\lambda A.\lambda B.\exists f[CH(f) \wedge \mathbf{some}'(f)(A)(B)]$ である. そして **some′** という determiner は次のように定式化されることを利用する. [15, p.111]

$$(20) \quad < >_{((et)e)((et)((et)t))} \stackrel{\text{def}}{=} \lambda g_{(et)e}.\lambda X_{et}.\lambda Y_{et}.X \neq \emptyset \wedge Y(g(X))$$

(19) の **E** は「通常の」存在量化を表す determiner である. [15, p.53] これにより (18) に対して, 次のような派生が可能になる.

$$
\dfrac{
\dfrac{
\dfrac{ECC \quad \text{some}'}{\mathbb{E} \qquad \text{man}'}
\qquad \text{admire}'
}{\mathbf{E}(\mathbf{man}') \qquad SAT(\mathbf{E}(\mathbf{man}'), \mathbf{admire}')}
\qquad
\dfrac{
\dfrac{ECC \quad \text{some}'}{\mathbb{E} \qquad \text{woman}'}
}{\mathbf{E}(\mathbf{woman}')}
}{\text{if}' \quad \mathbb{E}(\text{woman}') \quad (SAT(\mathbb{E}(\text{man}'), \text{admire}'))}
\qquad
\dfrac{\text{then}' \quad \text{ok}'}{\mathbf{then}'(\mathbf{ok}')}
}{\mathbf{if}'(\mathbf{E}(\mathbf{woman}')(SAT(\mathbf{E}(\mathbf{man}'), \mathbf{admire}')))(\mathbf{then}'(\mathbf{ok}'))}
$$

簡単に解説する. まず **E(woman′)** は (19) より次の様に展開される.

$$(\lambda A.\lambda B.A \cap B \neq \emptyset)\ (\mathbf{woman}')$$

一方 $SAT(\mathbf{E}(\mathbf{man}'), \mathbf{admire}')$ は (3.3) より以下となる.

$$\begin{aligned}
&\lambda y.\mathbf{E}(\mathbf{man}')(\lambda x.\mathbf{admire}'(x)(y)) \\
&= \lambda y.\lambda B.\exists z[\mathbf{man}(z) \wedge B(z)].[\lambda x.\mathbf{admire}'(x)(y)] \\
&= \lambda y.\exists z[\mathbf{man}' \wedge (\lambda x.\mathbf{admire}'(x)(y))(z)] \\
&= \lambda y.\exists z[\mathbf{man}'(z) \wedge \mathbf{admire}(z)(y)]
\end{aligned}$$

したがってこのふたつから以下が導かれる.

$$\begin{aligned}
&(\lambda B.\mathbf{woman}' \cap B \neq \emptyset)(\lambda y.\exists z[\mathbf{man}'(z) \wedge \mathbf{admire}(z)(y)]) \\
&= \exists x[\mathbf{woman}' \wedge [\lambda y.\exists z[\mathbf{man}' \wedge \mathbf{admire}'(z)(y)]](x)] \\
&= \exists x[\mathbf{woman}'(x) \wedge \exists z[\mathbf{man}'(z) \wedge \mathbf{admire}'(z)(x)]]
\end{aligned}$$

これに定理 $A \wedge \exists xB = \exists x(A \wedge B)$ を適用し, さらに条件文を加えた結果は, 通常の存在量化を伴った次の解釈に等しい.

$$[\exists x \exists y[\mathbf{woman}'(x) \wedge \mathbf{man}'(y) \wedge \mathbf{admire}'(y)(x)]] \rightarrow [\mathbf{ok}']$$

ここで indefinite は「島」の条件を守っている. すなわち条件文のスコープを越えていない.

これに対して, ふたつの indefinite が条件文のスコープを越えた派生は次のように可能となる. 準備として some woman admires some man を次のように導く.

$$\dfrac{\dfrac{\mathbf{some}' = \lambda A.\lambda B.\ <f>\ (A)(B), \quad \mathbf{woman}'}{\lambda B.\lambda\ <f>\ (\mathbf{woman}')(B)} \quad \dfrac{\mathbf{admire}', \quad \dfrac{\mathbf{some}', \ \mathbf{man}'}{\lambda B.\ <f>\ (\mathbf{man}')(B)}}{SAT(\lambda B.\ <f>\ (\mathbf{man}')(B), \mathbf{admire}')}}{\lambda g.\lambda f.(\lambda.B\ <f>\ (\mathbf{woman}')(B)(SAT(\lambda B.\ <g>\ (\mathbf{man}')(B), \mathbf{admire}'))}$$

上の派生を細かく見る.

右中段の $\mathbf{admire}', \lambda B.\ <f>\ (\mathbf{man}')(B) \Rightarrow SAT(\lambda B.\ <f>\ (\mathbf{man}')(B), \mathbf{admire}')$ については以下のように派生される. はじめに公理 (9) から

$$\mathbf{admire}' \Rightarrow \mathbf{admire}', \quad \lambda B.\ <g>\ (\mathbf{man}')(B) \Rightarrow \lambda B.\ <g>\ (\mathbf{man}')(B)$$

このふたつの仮定は他動詞 $e(et)$ と (単数の) 量化子 $(et)t$ であるから, SAT (3.3) を適用し, 以下が導かれる.

$$\mathbf{admire}', (\lambda B.\ <f>\ (\mathbf{man}')(B)) \Rightarrow (SAT(\lambda B.\ <f>\ (\mathbf{man}')(B), \mathbf{admire}'))$$

すなわち $e(et)$ と $(et)t$ から et を導く．この結果と，左中段の $\lambda B.\lambda <f>$ **(woman′)**(B) より以下を導く．$\lambda B.\ <f>$ **(woman′)**(B) は $\lambda f.\lambda B.\ <f>$ **(woman′)**$(B)(g)$ であるから，

$$\dfrac{\dfrac{\dfrac{\textbf{admire}',\ (\lambda f.\lambda B.\ <f>\ (\textbf{man}')(B))(g)}{\Rightarrow (\lambda f.SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}'))(g)}}{\textbf{admire}',\ \lambda f\lambda B.\ <f>\ (\textbf{woman}')(B) \Rightarrow \lambda f.SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}')} \quad e(et),\ (et)t \vdash et}{e(et),\ ((et)e) \to ((et)t) \vdash ((et)e) \to (et)}$$

となる．

この結果を some woman と合成し，t タイプを導出するわけだが

(21) $(\lambda B.\lambda <f>\ (\textbf{woman}')(B))(SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}'))$

ここでも ACOND を適用していく．はじめに量化子 (主語) に ACOND を適用し，$(et)e \to (et)t$ タイプの関数を導出し，これと述部を合成する．

$$\lambda f.\lambda B.\ <f>\ (\textbf{woman}')(B),\quad SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}') \Rightarrow \lambda f.\ <f>\ (\textbf{woman}')(SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}'))$$
$$(et)e \to (et)t,\ et \vdash ((et)e) \to t$$

次に述部に ACOND を適用する．

$$\lambda f.\lambda B.\ <f>\ (\textbf{woman}')(B),\quad \lambda g.(SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}')) \Rightarrow \lambda g.\lambda f.\ <f>\ (\textbf{woman}')(SAT(\lambda B.\ <f>\ (\textbf{man}')(B), \textbf{admire}'))$$
$$(et)e \to (et)t,\ ((et)e) \to et \vdash ((et)e) \to (((et)e) \to t)$$

これは $(et)e \to t$ のタイプである．この結果は $(et)e \to ((et)e \to t)$ となる．次に，この結果を φ と置き換えて，次の派生を導く．

(22)
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\textbf{if}'\ \varphi}{\lambda g.\lambda f.\textbf{if}'((\lambda B.\ <f>\ (\textbf{woman}')(B)} \quad \dfrac{\textbf{then}'\ \textbf{ok}'}{\textbf{then}'(\textbf{ok}')}}{(SAT(\lambda B.\ <g>\ (\textbf{man}')(B), \textbf{admire}'}}{ECC\quad \lambda g.\lambda f.\textbf{if}'((\lambda B.\ <f>\ (\textbf{woman}')(B)\ (SAT(\lambda B.\ <g>\ (\textbf{man}')(B), \textbf{admire}')))(\textbf{then}'(\textbf{ok}'))}}{ECC\quad SAT(ECC, \lambda g.\lambda f.\textbf{if}'((\lambda B.\ <f>\ (\textbf{woman}')(B))\ (SAT(\lambda B.\ <g>\ (\textbf{man}')(B), \textbf{admire}')))(\textbf{then}'(\textbf{ok}'))}$$
$$SAT(ECC, SAT(ECC, \lambda g.\lambda f.\textbf{if}'((\lambda B.\ <f>\ (\textbf{woman}')(B))\ (SAT(\lambda B.\ <g>\ (\textbf{man}')(B), \textbf{admire}')))(\textbf{then}')(\textbf{ok}')))$$

最後の式について補足する.

もっとも内側の式 ($SAT(\lambda B.\ <g>\ (\mathbf{man}')(B), \mathbf{admire}')$) から展開すると

$$\lambda x.\lambda B.\ <g>\ (\mathbf{man}')(B)(\lambda y.\mathbf{admire}'(y)(x))$$
$$= \lambda x.\ <g>\ (\mathbf{man}')(\lambda y.\mathbf{admire}'(y)(x))$$

これと $\lambda B.\ <f>\ (\mathbf{woman}')(B)$ を合成する.

(23) $$<f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda y.\mathbf{admire}'(y)(x)))$$

この結果に $\mathbf{if}' = \lambda\varphi.\lambda\psi.\varphi \rightarrow \psi$ を適用する.

$$<f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda y.\mathbf{admire}'(y)(x))) \rightarrow \mathbf{ok}'$$

この結果と ECC (3.4) $= \lambda A.A \cap CH \neq \emptyset$ に SAT を適用する.

$$SAT(ECC,\ (\lambda g.\lambda f.\ <f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda y.\mathbf{admire}'(y)(x))) \rightarrow \mathbf{ok}'))$$
$$= \lambda f.\lambda A_{(et)e}.A \cap CH \neq \emptyset\ (\lambda g.\ <f>\ (\mathbf{woman}')$$
$$(\lambda x.\ <g>\ (\mathbf{man}')(\lambda y.\mathbf{admire}'(y)(x))) \rightarrow \mathbf{ok}')$$
$$= \lambda f.(\lambda g.\ <f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda g.\mathbf{admire}'(y)(x))) \rightarrow \mathbf{ok}') \cap CH \neq \emptyset$$

最後に一番外側の SAT を適用する.

$$= SAT(ECC,\ \lambda f.(\lambda g.\ <f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda g.\mathbf{admire}'(y)(x)))$$
$$\rightarrow \mathbf{ok}') \cap CH \neq \emptyset)$$
$$= \lambda g.\lambda A.A \cap CH \neq \emptyset(\lambda f.\ <f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda g.\mathbf{admire}'(y)(x)))$$
$$\rightarrow \mathbf{ok}') \cap CH \neq \emptyset)$$
$$= \lambda g.(\lambda f.\ <f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda g.\mathbf{admire}'(y)(x)))$$
$$\rightarrow \mathbf{ok}') \cap CH \neq \emptyset) \cap CH \neq \emptyset$$

この結果は, ふたつの選択関数 f, g と, 選択関数全体の集合 CH の積が空集合ではないことを表している. つまりこの式の条件を充たすようなふたつの関数 f, g がかならず存在することを意味している. したがって次の式と同一である.

$$\exists f\,\exists g[CH(f) \wedge CH(g) \wedge [<f>\ (\mathbf{woman}')(\lambda x.\ <g>\ (\mathbf{man}')(\lambda y.\mathbf{admire}'(y)(x)))$$
$$\rightarrow \mathbf{ok}']]$$

これにより, 選択関数を導入した存在量化のシステムが, 通常の述語論理と同じ結果をもたらすことが示された.

参考文献

[1] 赤間世紀. 自然言語・意味論・論理. 共立出版, 1998.

[2] Jon Barwise and Robin Cooper. Generalized Quantifier and Natural Language. *Linguistics and Philosophy*, Vol. 4, No. 3, pp. 159–219, 1981.

[3] David R. Dowty, Rober E. Wall, and Stanley Peters. *Introduction to Montague Semantics*. D. Reidel, 1981.

[4] Paul Halmos and Steven Givant. *Logic as Algebra*. The Mathematical Association of America, 1998.

[5] Edward L. Keenan and Leonald M. Faltz. *Boolean Semantics for Natural Language*. D. Reidel, 1985.

[6] 松本和夫. 復刊・数理論理学. 共立出版, 2001.

[7] 小野寛晰. 情報科学における論理. 日本評論社, 1994.

[8] Barbara H. Partee, Alice ter Meulen, and Robert E. Wall. *Mathematical Methods in Linguistics*. Kluwer Academic Publishers, 1981.

[9] Tanya Reinhart. Quantifier Scope: How Labor is Divied between QR and Choice Funcitons. *Linguistics and Philosophy*, Vol. 20, No. 4, pp. 335–397, 1998.

[10] 清水義夫. 記号論理学. 東京大学, 1984.

[11] 白井賢一郎. 形式意味論入門. 産業図書, 1985.

[12] 白井賢一郎. 自然言語の意味論. 産業図書, 1991.

[13] 竹内外史. 現代集合論入門 [増補版]. 日本評論社, 1989.

[14] Johan van Benthem. *Language in Action: Categories, Lambdas, and Dynamic Logic*. The MIT Press, 1995.

[15] Yoad Winter. *Flexibility Principles in Boolean Semantics: The interpretation of Coordination, Plularity, and Scope in Natural Language*. The MIT Press, 2001.