Vol.2013-NL-214 No.19
2013/11/15

限定継続を用いた only のフォーカスの分析と実装に向けて

叢 悠悠[1,a)]　浅井 健一[2,b)]　戸次 大介[2,3,4,c)]

概要：プログラミングにおける「継続」とは，残りの計算，すなわちある部分項に対する文脈のことを指す．この概念を自然言語の意味論に取り入れることで，様々な言語現象の意味を記述することができる．本研究では，限定継続命令 shift/reset を用いた副詞 only のフォーカスの分析 (Bekki and Asai (2010)) を OCaml で実装し，一つのフォーカスを含む文の意味表示を正しく計算できることを確認した．しかし，フォーカスが複数存在する場合への非対応など，今回の実装にはいくつかの問題があり，それらについても考察する．

キーワード：フォーカス，only，限定継続，shift/reset

Towards the Analysis and the Implementation of Focus on "only" with Delimited Continuations

Abstract: Continuations in programming languages represent the rest of computation, i.e., the surroundings of a given subterm. Applying this notion to natural language semantics enables us to analyze various linguistic phenomena. In this paper, we will present an implementation of the analysis of focus on "only" with control operators *shift* and *reset* (Bekki and Asai (2010)), which can correctly calculate the meaning of sentences which contain one focus. However, there are several problematic phenomena such as sentences with more than two foci. We will also discuss these problems.

Keywords: focus, only, delimited continuations, shift/reset

1. はじめに

自然言語で記述されたテキストの意味を形式化する際に，特定の語彙項目がそれを取り囲むコンテキストを参照することを要求する言語現象が存在する．これらの現象は従来，非顕在的移動 (covert movement) を起こすことによって，正しい意味計算が可能になるとされてきた．しかし，このような手法は，移動を許さない範疇文法やランベックスタイルの文法に組み込むことができない．

一方で近年，移動を伴う言語現象を限定継続 (delimited continuation) で分析する研究が行われている (Shan [10], Barker [1], Barker [2], Shan and Barker [11], Barker and Shan [3])．継続とは，プログラミングにおいて「残りの計算」を表す概念である．自然言語において，あるフレーズに対するコンテキストは継続と見なすことができるのである．

本研究では限定継続を用いた *only* のフォーカスの分析を実装した．以下，第 2 節で継続について解説する．第 3 節では非顕在的移動によって分析されてきた言語現象の一例として，フォーカスに関する先行研究を取り上げ，第 4 節で Bekki and Asai [4] の分析の実装について述べる．第 5 節・第 6 節では Bekki and Asai [4] の分析の問題点と今後の課題について論じる．

[1] お茶の水女子大学
Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo, 112–8610, Japan
[2] お茶の水女子大学大学院人間文化創成科学研究科
Ochanomizu University, Graduate School of Humanities and Science, 2-1-1 Otsuka, Bunkyo-ku, Tokyo, 112–8610, Japan
[3] 国立情報学研究所
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
[4] 独立行政法人科学技術振興機構, CREST
CREST, Japan Science and Technology Agency, 4-1-8 Honcho, Kawaguchi, Saitama, 332-0012, Japan
[a)] g1020519@is.ocha.ac.jp
[b)] asai@is.ocha.ac.jp
[c)] bekki@is.ocha.ac.jp

2. 限定継続

プログラミングにおいて，継続とは計算のある時点における残りの計算のことを指す．例えば，$(3*2)+1$ という計算の $3*2$ の部分を実行しているときの継続は，「現在行っている計算の結果に 1 を足す」という関数になる．

限定継続は，継続を考慮する範囲を限定したものである．限定継続を扱うための命令にはさまざまなものがあるが，本稿では shift/reset (Danvy and Filinski [5]) を採用する．shift はその時点の継続を関数として切り取る命令，reset は継続を考慮する範囲を限定する命令である．たとえば，$1 + \mathit{reset}\,(2 * (\mathit{shift}\,k.\,k\,3))$ の k には $f(x) = 2 * x$ という関数が入り，全体の計算結果は 7 になる．shift 節内に k が複数回現れることも許される．また，k が現れない場合，継続は破棄される．$1 + \mathit{reset}\,(2 * (\mathit{shift}\,k.\,k\,(k\,3)))$ は 13 になり，$1 + \mathit{reset}\,(2 * (\mathit{shift}\,k.\,3))$ は 4 となる．

直接形式のプログラムにおいて，継続は陽に表れていないため，継続を取り出したい場合はプログラムを継続渡し形式 (Continuation Passing Style) に変換する必要がある．以下に Plotkin [7] の CPS 変換規則を示す[*1]．

$$
\begin{aligned}
[\![x]\!] &= \lambda k.\,k\,x \\
[\![\lambda x.\,M]\!] &= \lambda k.\,k\,\lambda x.\,[\![M]\!] \\
[\![M\,N]\!] &= \lambda k.\,[\![M]\!]\lambda m.[\![N]\!]\lambda n.\,m\,n\,k
\end{aligned}
$$

継続渡し形式のプログラムは，各関数に継続のための引数が追加されることによって，継続が明示的に表れている．shift/reset オペレータは継続渡し形式で以下のように定義される：

$$
\begin{aligned}
[\![\mathit{shift}\,c.\,M]\!] &= \lambda k.[\![M]\!][\lambda v.\lambda k'.k'\,(k\,v)/c]\,\lambda x.x \\
[\![\mathit{reset}\,(M)]\!] &= \lambda k.k\,([\![M]\!]\,\lambda x.x)
\end{aligned}
$$

shift/reset オペレータを使うと，直接形式のプログラムで継続を扱うことが可能になる．そして，与えられたプログラムを継続渡し形式に変換して解釈するインタプリタを用意すれば，shift/reset を含むプログラムを評価することができる．

3. フォーカスに関する先行研究

3.1 Alternative semantics

John only loves Mary という文について，Mary が *only* のフォーカスとなっているとき，この文が真となるのは John が他の誰でもなく Mary だけを愛している場合である．もし，John が愛している Mary でない人物が存在すれば，この文は偽となる．このような現象を説明するために，Rooth は以下のような alternative semantics を提案した．

Rooth [9] は，フォーカスを含む文 α は通常の semantic value $[\![\alpha]\!]^o$ に加えて，フォーカスが意味に与える影響を考慮するための focus semantic value $[\![\alpha]\!]^f$ をもつとしている．Focus semantic value はフォーカス部分に同じ型をもつ他の語句を代入した命題の集合であり，以下の定義に従って再帰的・構成的に得られる：

a. semantic type τ のフォーカスされたフレーズの focus semantic value は，τ 型の可能な denotation の集合である．
b. フォーカスされていない語彙項目の focus semantic value は，通常の semantic value からなる単一集合である．
c. α が α_1, ..., α_k からなるフォーカスされていない複合的なフレーズで，Φ が α に対する意味論的規則（たとえば関数適用）のとき，α の focus semantic value は $\{\Phi(x_1, ..., x_k) \mid x_1 \in [\![\alpha_1]\!]^f \wedge ... \wedge x_k \in [\![\alpha_k]\!]^f\}$ なる集合である．

このようにして得られる集合を alternative set とよぶ．この集合の要素は $[\![\alpha]\!]^o$ の比較対象となっている．以下，フォーカスされた語句を $[\ ]_F$ で表すとし，(1) の文を考える．

(1) John loves $[\text{Mary}]_\text{F}$
$[\![\text{John loves } [\text{Mary}]_\text{F}]\!]^f = \{\text{John loves x} \mid \text{x} \in C\}$[*2]
$[\![\text{John loves } [\text{Mary}]_\text{F}]\!]^o = \{\text{John loves Mary}\}$

(1) の focus semantic value は $[\text{Mary}]_\text{F}$ の部分が抽象化された命題の集合になる．フォーカスを含む文を解釈するためには，このような alternative set を考慮する必要がある．文脈に応じて alternative set を制限するためのオペレータとして，Rooth [9] は $\sim$ を以下のように定義している：

(2) ϕ が統語的に正しいフレーズで，C が文脈によって定められたドメインのとき，$\phi \sim C$ は C が $[\![\phi]\!]^f$ の部分集合で，$[\![\phi]\!]^o$ と一つ以上の他の要素を含む，という前提を導入する．

[*1] この関数適用に対する変換規則は値呼び戦略 (call-by-value) の左から順に実行する場合のものである．右から実行する場合は以下のようになる：
$[\![MN]\!] = \lambda k.\,[\![N]\!]\lambda n.\,[\![M]\!]\lambda m.\,m\,n\,k$

[*2] ここで，C はフォーカスに対する alternatives の集合である．Rooth (1997) では，このように量化のドメインを文脈に応じて定めることで，*only* の全称量化で考慮する集合を制限している．

さらに，*only* に対する意味表示は以下の通りである：

(3) $[\![\text{only}]\!] = \lambda C \lambda p \forall q\,[((q \in C) \wedge {}^{\vee}q) \leftrightarrow (q = p)]$ [*3]

これらの定義と (1) をあわせて，以下の表示を考える．

(4) $[_{\text{S}}\ \text{only}(C)\ [_{\text{S}}[_{\text{S}}\ \text{John loves Mary}_{\text{F}}] \sim C]]$

(4) の場合，(3) の p は John loves Mary という命題であり，フォーカス部分にある語句を代入した命題 q が成り立つのは，その語句が Mary であるとき，またそのときのみとされる．

3.2 フォーカスの移動

Wagner [13] では，動詞句に接続する *only* がそのフォーカスと結びつくために，フォーカスを *only* の補部へと移動させている（**図 1**）．この移動は非顕在的移動の一種である．移動によって，一度構築された構文木は構造の異なる木へと変化する．生成文法では，構文木が音声形式と論理形式に分離した後にこのような移動を起こすことがある．一方，各語彙項目から局所的に意味を計算する範疇文法は移動を許さず，非顕在的移動が必要となる場合は型の不整合が生じる．これは範疇文法の欠点の一つである．しかし，Wh 移動と異なり，非顕在的移動はどのような場合に移動を起こすか，また，どこへ移動するかを統語的な構造から決定することができない．そのため，非顕在的移動を含む分析を実装した場合，その計算量は大きくなると予測される．さらに，移動は可能であるときに起こすものとされており，その停止性については議論されていない．

本研究では限定継続を用いることで非顕在的移動を起こさずに意味を計算する．継続を扱うためには CPS 変換を行うが，実は CPS 変換も木から木への変換として考えられる．ただし，CPS 変換については停止性を議論することが可能である．なぜなら，任意のプログラムは部分式の CPS 変換に分解され，定義に従って再帰的に変換をしていくと，必ずベースケースに帰着するためである．したがって，CPS 変換は非顕在的移動と比べて計算的に良い性質をもっているといえる．

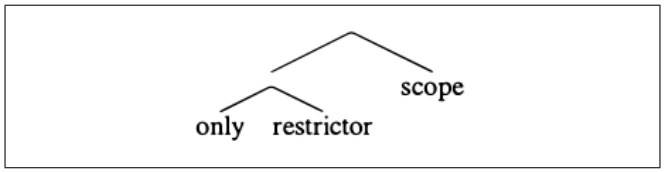

図 1 フォーカスの移動
Fig. 1 Focus Movement

[*3] ${}^{\vee}q$ は q が成り立つことを表す．

3.3 fcontrol/run を用いた分析

Barker [2] では，限定継続命令 fcontrol/run (Sitaram [12]) を用いたフォーカスの分析を行っている．run は継続を限定するための命令であり，0 引数関数 (thunk) とハンドラとよばれる手続きを引数として受け取る．fcontrol は引数を一つ要求し，その引数および最も近い run までの継続をハンドラに渡す．例として $(1 + (\mathit{run}\,(2 * ((\mathit{fcontrol}\,3) + 4))\,(\lambda x k.\,k\,(k\,x))))$ を考えると，run で切り取られる継続 k は $\lambda y.\,(2 * (y + 4))$ であり，この関数に fcontrol の引数 3 が渡されて，最終的な計算結果は 37 となる．

Barker [2] は Lisp 系言語 Scheme のスタイルにしたがって，フォーカスを fcontrol，*only* を run で表している．以下に *only* と John only drinks $[\text{Perrier}]_{\text{F}}$ に対する意味表示を示す．

(5) $[M]_{\text{F}} = \mathit{fcontrol}\,M$
$[\![\mathit{only}\,P]\!] = \mathit{run}P\lambda x k.\,(\mathit{and}\,(k\,x)\ (\forall z\,(\mathit{or}\,(\mathit{equal}\,x\,z)\,(\mathit{not}\,(k\,z)))))$

(6) John only drinks $[\text{Perrier}]_{\text{F}}$
$[\![(\mathit{run}\,(\mathit{drinks}\,(\mathit{fcontrol}\,\mathit{Perrier})\,j)\ (\mathit{lambda}\,(x\,k)\,(\mathit{and}\,(k\,x)\ (\forall z\,(\mathit{or}\,(\mathit{equal}\,x\,z)\,(\mathit{not}\,(k\,z))))))]\!]$
$= (\mathit{and}\,(\mathit{drinks}\,j\,\mathit{Perrier})\ (\forall z\,(\mathit{or}\,(\mathit{equal}\,\mathit{Perrier}\,z)\ (\mathit{not}\,(\mathit{drinks}\,z\,j)))))$

(6) の場合，*drinks* の第一引数に $(\mathit{fcontrol}\,\mathit{Perrier})$ が渡されることによって，継続 k は $\lambda x.\,\mathit{drinks}\,x\,j$ という関数になる．この k にフォーカスと同じ型をもつ語句を代入することで alternative set が得られる．

3.4 shift/reset を用いた分析

Bekki and Asai [4] では，shift/reset オペレータを使うことで移動を起こさずにフォーカスの意味計算を行っている．フォーカスと *only* に対する定義は以下の通りである：

$$[M]_F \stackrel{\text{def}}{\equiv} \mathit{shift}\,k.\forall x\,(kx \leftrightarrow x = M)$$
$$\mathit{only}\,(\phi) \stackrel{\text{def}}{\equiv} \mathit{reset}\,(\phi)$$

フォーカスを shift オペレータ，*only* を reset オペレータで表現することにより，継続 k はフォーカス部分が抽象化された命題となる．

Mary only introduced Bill to Sue という文について，フォーカスを変えた 2 種類の意味表示を (7), (8) に示す．

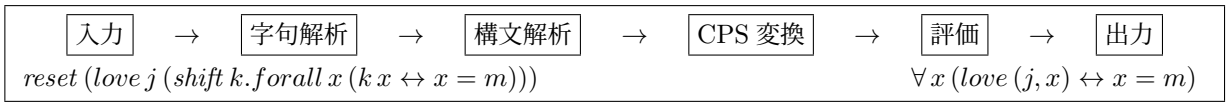

図 2 処理の流れ
Fig. 2 Procedure

(7) Mary only introduced $[\text{Bill}]_{\text{F}}$ to Sue

$$\begin{aligned}&[\![reset\,(introduce(m,\,(shift\,k.\forall x\,(k\,x \leftrightarrow x = b)),\,s))]\!]\\ &= [\![reset\,(introduce\,(m,\,[b]_F,\,s) \leftrightarrow x = b)]\!]\\ &= \forall x\,(introduce\,(m,\,x,\,s) \leftrightarrow x = b)\end{aligned}$$

(8) Mary only introduced Bill to $[\text{Sue}]_{\text{F}}$

$$\begin{aligned}&[\![reset\,(introduce(m,\,b,\,(shift\,k.\forall x\,(k\,x \leftrightarrow x = s))))]\!]\\ &= [\![reset\,(introduce\,(m,\,b,\,[s]_F) \leftrightarrow x = s)]\!]\\ &= \forall x\,(introduce\,(m,\,b,\,x) \leftrightarrow x = s)\end{aligned}$$

4. 実装

本研究では，Bekki and Asai [4] の定義に従った意味表示を解釈するインタプリタを関数型言語 OCaml で実装した．

対象言語（解釈される言語）は単純型付きラムダ計算の体系に論理演算子 $=, \leftrightarrow, \wedge, \neg, \forall, \exists$ と *shift*, *reset* および二項述語 *love*，三項述語 *introduce* を追加したものである（**図 3**）．*love* と *introduce* はそれぞれ関数として定義してあり，たとえば *love* は引数 x, y を受け取ったらコンストラクタ $Love(x, y)$ を返す．

$$\begin{aligned}E \;=\; &x \mid c \mid E = E \mid E \leftrightarrow E \mid E \wedge E \mid \neg E\\ &\mid love\,E\,E \mid introduce\,E\,E\,E\\ &\mid forall\,x\,(E) \mid exists\,x\,(E) \mid shift\,k.\,E \mid reset\,(E)\end{aligned}$$

図 3 対象言語
Fig. 3 Object Language

入力された意味表示は字句解析・構文解析を経たのち，継続渡し形式に変換される．shift/reset の部分を正しく計算できるような入力であれば，簡約後の意味表示はこれらのオペレータを含まないものになる（**図 2**）．

以下に実行例を示す．**図 4**, **図 5** は前節の例文 (7), (8), **図 6** はフォーカスが動詞句にある場合，**図 7** はフォーカスを伴わない場合である[*4]．

```
$ ./interpreter
reset (introduce m (shift k. (forall x (k x <-> x = b))) s)
Answer: ∀ x((introduce (m, x, s) <-> (x = b)))
```

図 4 実行例 1：Mary only introduced $[\text{Bill}]_{\text{F}}$ to Sue
Fig. 4 ex.1 : Mary only introduced $[\text{Bill}]_{\text{F}}$ to Sue

[*4] なお，ここでは始めから意味表示を入力として与えているが，将来的には自然言語で記述されたテキストをそのまま入力し，構文解析をしたのち意味計算を行うように改良したいと考えている．

```
$ ./interpreter
reset (introduce m b (shift k. (forall x (k x <-> x = s))))
Answer: ∀ x((introduce (m, b, x) <-> (x = s)))
```

図 5 実行例 2：Mary only introduced Bill to $[\text{Sue}]_{\text{F}}$
Fig. 5 ex.2 : Mary only introduced Bill to $[\text{Sue}]_{\text{F}}$

```
$ ./interpreter
reset ((shift k. (forall p (k p <-> p = introduce))) m b s)
Answer: ∀ p((p m b s) <-> (p = introduce))
```

図 6 実行例 3：Mary only $[\text{introduced}]_{\text{F}}$ Bill to Sue
Fig. 6 ex.3 : Mary only $[\text{introduced}]_{\text{F}}$ Bill to Sue

```
$ ./interpreter
reset (introduce m b s)
Answer: introduce (m, b, s)
```

図 7 実行例 4：Mary only introduced Bill to Sue
Fig. 7 ex.4 : Mary only introduced Bill to Sue

フォーカスが動詞にある場合は，名詞句にある場合と異なる扱いをする必要がある．図 4 に示した例において，捕捉される継続 k は $\lambda y.\,introduce\,(m,\,y,\,s)$ という関数であり，これに全称量化子が束縛する変数 x が適用される．一方，図 6 の例では，全称量化された変数 p が述語，すなわち関数であることを想定している．この場合，継続 k は p に対する関数適用であり，$k\,p$ は $p\,m\,b\,s$ という計算となるが，p は自身が関数であるという情報をもたない．そのため，通常の関数適用として解釈すると，引数を渡す際にエラーが生じてしまう．これを回避するために，今回の実装では単なる変数に対する関数適用のケースを設けている．

5. 問題点

本節では Bekki and Asai [4] の分析と今回の実装の問題点について考察する．

5.1 Multiple focus への対応

今回作成したインタプリタは，複数のフォーカスが存在する文に対して正しく意味を計算することができない．Mary only introduced $[\text{Sue}]_F$ to $[\text{Bill}]_F$ を左から解釈すると以下のようになる：

$$\begin{aligned}&[\![reset\,(introduce\,m\,(shift\,k.\forall x\,(k\,x \leftrightarrow x = s))\\ &\qquad\qquad (shift\,k'.\forall y\,(k'\,y \leftrightarrow y = b)))]\!]\\ &= \forall x\,((\forall y\,((introduce\,(m,\,x,\,y) \leftrightarrow (y = b))) \leftrightarrow (x = s)))\end{aligned}$$

しかし，期待される意味表示は以下の通りである：

$$\forall x\,(\forall y\,(introduce\,(m,\,x,\,y) \leftrightarrow ((x = s) \wedge (y = b))))$$

この文を正しく解釈するためには，単に前から順番に計算するのではなく，一つの shift 命令で複数箇所を抽象化する必要がある．

一方，偶然正しい意味表示が得られる場合もある．Rooth [9] に以下のような例文がある：

(9) John only introduced $[\text{Bill}]_{\text{F}}$ to Mary.
He also only introduced $[\text{Bill}]_{\text{F}}$ to $[\text{Sue}]_{\text{F}}$.

2 文目の $[\text{Sue}]_{\text{F}}$ は *also*，$[\text{Bill}]_{\text{F}}$ は *only* に対するフォーカスである．文全体としては，also が最も高いスコープをとっている．*also* のフォーカスが $shift\,k.\exists x\,(k\,x \wedge \neg(x = M))$ という前提を導入すると考えて，この文を右から解釈すると，以下の意味表示が得られる：

$$\begin{aligned}&[\![reset\,(introduce\,m\,(shift\,k.\forall x\,(k\,x \leftrightarrow x = b))\\ &\qquad\qquad (shift\,k'.\exists y\,(k'\,y \wedge \neg(y = s))))]\!]\\ &= \exists y\,((\forall x\,((introduce\,(j,\,x,\,y) \leftrightarrow (x = b))) \wedge (\neg(y = s))))\end{aligned}$$

この場合，*also* が外側のスコープをとっているため，*only* のフォーカスの意味表示の後ろに $\neg(y = s)$ が *also* の意味表示に含まれる $\wedge$ オペレータで結合している．しかし，同じ文でも *also* のフォーカスを $[\text{Bill}]_{\text{F}}$，*only* のフォーカスを $[\text{Sue}]_{\text{F}}$ とする読みを考えると，上と同様の問題が生じる．

5.2 ネストしたフォーカスの記述

Rooth [9] による focus semantic value の定義に従うと，全てのフォーカスがフォーカス自身と最も近いオペレータと結合してしまう．一方，Krifka [6] では，一つのフォーカスを束縛するオペレータが複数存在する場合に，ネストしたフォーカスを用いてこの問題を回避している．

(10) Last month John only drank $[\text{beer}]_{\text{F}}$
He has also only drunk $[[\text{wine}]_{\text{F}}]_{\text{F}}$

2 文目について，外側のフォーカスが *only* に，内側が *also* によって束縛されている読みは，以下の論理形式で表される：

(11) also $[_{\text{S}}$ $[\text{wine}]_{\text{F}}$ λe_2 $[_{\text{S}}$ have $[_{\text{S}}$ only $[_{\text{S}}$ $[e_2]_{\text{F}}$ λe_1 $[_{\text{S}}$ John drunk $e_1]]]]]$

フォーカスを shift オペレータ，*only* や *also* を reset オペレータで表現するという本研究の考えに基づくと，ネストしたフォーカスは shift 命令と reset 命令を 2 回実行する必要があると考えられる．(10) の場合，*also* に対する reset は *only* に対するものの外側にある．一方，*also* に対する shift は内側にあるため，この shift は *only* に対する reset を越えて継続を切り取る必要がある．しかしこれは許されず，shift で切り取られるのは内側の reset までの継続となる．外側の継続を取ってくるためには，階層づけをした継続 (Danvy and Filinski [5]) を用いるか，あるいは CPS 変換を 2 回実行することで，レベルの異なる継続を区別するという方法が考えられる．

5.3 計算体系の拡張

もう一つの問題は，拡張した意味表示の体系にある．カリー＝ハワード同型対応により，単純型付きラムダ計算と命題論理の間には厳密な対応関係がある．たとえば，λ 演算子は $\rightarrow$ の証明と対応づけられる．しかし，単純型付きラムダ計算には，全称量化および存在量化に対応する構文が存在しない．そのため，今回の実装ではそれらに相当するコンストラクタを設けたが，β 簡約/η 簡約のペアで意味が定められている λ 演算子と異なり，量化子を含む項については，その振る舞いを簡約規則で定義することができない．この観点から，拡張した体系では性質の異なる構文を同列に扱っているといえる．さらに，量化子によって束縛される変数は型ではなく項である．つまり，量化を含む計算体系に拡張するためには，項を含む型を許す必要がある．

これらの問題を回避するために，筆者らは依存型を許す計算体系で分析・実装を行うことを考えている．依存型とは，項に依存する型のことである．たとえば，$\Pi x : A.B$ は A 型の引数を受け取ったら B 型の項を返す関数の型である．ここで，B は x を含み得るため，項 x に依存しているといえる．この Π 演算子は依存型理論において全称量化子に相当する演算子である．この体系において，$\rightarrow$ と $\forall$ および $\exists$ はともに導入規則と除去規則で意味が与えられている．したがって，これらの構文を同じように扱うのは至って自然である．

依存型を含むラムダ計算の体系において，John only loves $[\text{Mary}]_{\text{F}}$ の意味表示は以下のようになる：

$$P : reset\,(love\,m\,(shift\,k \rightarrow \Pi x : e.\,kx\ \rightarrow equal\,(x, j)))$$

ここで，P は *reset* 以下の型をもつ項である．この P は proof term とよばれ，ある特定の型をもつ項が存在することを意味している．

しかし，依存型に対する「継続」の概念は自明ではない．なぜなら，型の中に計算を含む依存型においては，項レベルと型レベルのコンテキストが異なるためである．実際，依存型をもつ体系に完全に対応する CPS 変換はまだ提案されておらず，筆者らは今後定式化を目指したいと考えている．

6. まとめと今後の課題

本研究では, Bekki and Asai [4] に従って，限定継続を使った *only* のフォーカスの意味計算を行うインタプリタを作成した．実装を行ったことによって，複雑な文に対して現在の理論が提示する予測を即座に得られるようになった.

しかし，前節で述べたように，今回の実装で正しく意味表示を計算できる場合は限られており，複数のフォーカスを含む文や，Krifka のネストしたフォーカスについては，さらなる分析が必要である．また，依存型理論への継続の導入も今後の課題である.

参考文献

[1] Barker, C.: Continuations and the Nature of Quantification, *Natural Language Semantics* 10(3), pp. 211–241 (2002).
[2] Barker, C.: Continuations in Natural Language, *the Fourth ACM SIGPLAN Continuations Workshop (CW'04). Technical Report CSR-04-1, School of Computer Science, University of Birmingham, Birmingham B152TT*, pp. 1–11 (2004).
[3] Barker, C. and Shan, C.-c: Types as Graphs: Continuations in Type Logical Grammar, *Journal of Logic, Language and Information* 15(4), pp. 331–370 (2006).
[4] Bekki, D. and K. Asai: Representing Covert Movements by Delimited Continuations, In: K. Nakakoji, Y. Murakami, and E. McCready (eds.): *New Frontiers in Artificial Intelligence (JSAI-isAI 2009 Workshops, Tokyo, Japan, November 2009, Selected Papers from LENLS 6)*, Vol. LNAI 6284, pp. 161–180 (2010).
[5] Danvy, O. and A. Filinski, Abstracting Control, In: *LFP90, the 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160 (1990).
[6] Krifka, M.: A Compositional Semantics for Multiple Focus Constructions, In: Moore, S. and Wyner, A. Z. (eds.), In: *Proceedings from Semantics and Linguistic Theory 1*, Cornell Working Papers in Linguistics 10, pp. 127–158 (1991).
[7] Plotkin, G. D.: Call-by-Name, Call-by-Value and Lambda Calculus, *Theoretical Computer Science* 1, pp. 125–159 (1975).
[8] Rooth, M.: A Theory of Focus Interpretation, *Natural Language Semantics* 1, pp. 75–116 (1992).
[9] Rooth, M.: Focus, *The Handbook of Contemporary Semantic Theory*, pp. 271–298 (1997).
[10] Shan, C.-c: A continuation semantics of interrogatives that accounts for Baker's ambiguity, In: B.Jackson (ed.): *Semantics and Linguistic Theory XII*, Cornell University Press, pp. 246–265 (2002).
[11] Shan, C.-c. and Barker, C.: Explaining Crossover and Superiority as Left-to-right Evaluation, *Linguistics and Philosophy*, 29(1), pp. 91–134 (2006).
[12] Sitaram, D.: Handling Control, In: *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM Press, pp. 147–155 (1993).
[13] Wagner, M.: NPI-Licensing and Focus Movement, In: E.Georgala and J.Howell (eds.): *Proceedings of SALT XV*. Ithaca, NY: CLC Publications, pp. 276–293 (2006).