限定継続を用いたフォーカスの分析と実装

叢悠悠 [1], 浅井健一 [2], 戸次大介 [3]

[1] お茶の水女子大学理学部情報科学科
g1020519@is.ocha.ac.jp

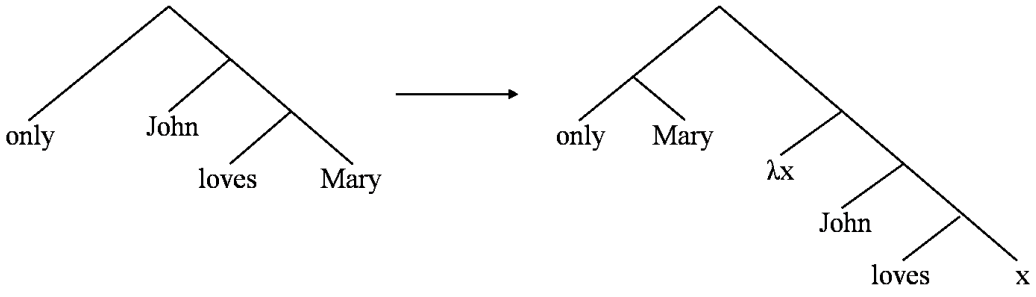
[2] お茶の水女子大学大学院人間文化創成科学研究科
asai@is.ocha.ac.jp

[3] お茶の水女子大学大学院人間文化創成科学研究科, 国立情報学研究所,
独立行政法人科学技術振興機構, CREST
bekki@is.ocha.ac.jp

概 要 フォーカスとよばれる言語現象においては，文の意味表示を与える際にフォーカスされた言語表現を取り囲むコンテキストが必要となる．一方，プログラミングの分野には，ある項に対する残りの計算を指す「継続」という概念がある．この継続は，フォーカスの周りのコンテキストとみなすことができる．Bekki and Asai (2010) は限定継続命令 shift/reset (Danvy and Filinski, 1990) を用いて “only” に対するフォーカスの意味表示を与えており，叢ら (2013) は Bekki らの定義に従った意味表示を簡約するインタプリタを実装したが，複数の副詞にフォーカスされた言語表現が存在する場合に，意味表示を正しく簡約できないという問題があった．本研究では，複数のフォーカスが複数のコンテキストに対応づけられると主張する．Caml Light に shift/reset を直接実装した言語 OchaCaml で二つの階層の限定継続を定義し，実際にそれらを用いて問題であった例文の意味表示を与えた結果，期待された形に簡約されることを示す．

1 はじめに

自然言語で記述された文を形式化する際，テキストは構文解析を経て，意味表示とよばれる論理式に変換される．そのなかで，フォーカスや逆スコープ[1]といった言語現象においては，意味表示を与える際に特定の言語表現を取り囲むコンテキストが必要となる．このコンテキストを取り出すために，これらの現象は非顕在的移動 (covert movement) を要求するとされてきた (Wagner [23], May [12])．非顕在的移動とは，文の表層的な構造から得られた構文木から，別の形の構文木に変換する操作である (May [13])．たとえば，“John only loves Mary” という文において，Mary が副詞 *only* のフォーカスとなっているとき，その構文木は以下のように変換される (Wagner [23])．



[1]複数の量化表現を含む文において，量化子のスコープ関係が表層的な順序と逆転する現象のことを指す．たとえば，”Someone loves everyone” という文には,「ある特定の人物 A が全ての人間を愛している」という読みのほかに,「全ての人間について，その人を愛する人物が存在する」という読みをもつ．この二つ目の読みを逆スコープ読みという．

左側の構文木を右側の構文木に変換することによって，表層的には不明確であった *only* のフォーカスが Mary であることが明らかになる．Mary が移動したあとの構文木において，右側の部分木は “John loves x” という文になる．これは Mary に対するコンテキストである．このように，非顕在的移動を起こすことで，フォーカスの周りのコンテキストを取り出すことができる．

一方で，プログラミングの分野には「継続」という概念がある．継続とは，ある部分項に対する残りの計算のことを指す．自然言語におけるコンテキストは継続とみなすことができる．実際に，継続を自然言語の意味論に応用する研究がされ始めており，さまざまな言語現象に対して継続を用いた分析が行われている (Barker [1], Barker [2], Bekki and Asai [3], Shan [21])．

本研究では，限定継続命令 `shift/reset` (Danvy and Filinski [5]) を用いて，フォーカスを含む文の意味表示を与える．また，OchaCaml (Masuko and Asai [11]) を使って二つの階層の限定継続命令を定義し，複数の副詞を含む文において，それぞれのフォーカスに対応するコンテキストの区別を図る．本論文では以下，第 2 節でフォーカス，第 3 節で継続について解説し，第 4 節で限定継続を用いたフォーカスの分析を取り上げる．第 5 節では筆者らによる先行研究とその問題点について述べ，第 6 節で OchaCaml を用いた分析と実装について述べる．

2 フォーカス

本節では，筆者らのアプローチとの比較のため，形式意味論の分野におけるフォーカスの先行研究として，Rooth [18] の分析を紹介する．

フォーカスとは，文のなかで新しい情報，あるいは重要な情報として強調される部分のことである．以下，フォーカスとなっている言語表現を $[\,]_{\mathrm{F}}$ で表すとし，“Mary only introduced Bill to Sue” について，副詞 *only* のフォーカスを変えた二つの文を考える．

(1) a. Mary only introduced $[\text{Bill}]_{\mathrm{F}}$ to Sue.
b. Mary only introduced Bill to $[\text{Sue}]_{\mathrm{F}}$.

(1a) は「Mary は Bill だけを Sue に紹介した」，(1b) は「Mary は Bill を Sue だけに紹介した」という意味である．これらの文に対して意味表示を与えることを考える．意味表示は文の意味的な構造を表す論理式であり，与えられた状況によってその真偽値が決定する．元の文が示している状況では真，そうでないときは偽となるような意味表示に変換するためには，その文の真理条件を考える必要がある．(1a) と (1b) は，*only* のフォーカスが Mary の紹介した人と，Mary が誰かを紹介した先の人のどちらになるかによって，真理条件が異なっている．たとえば，Mary が Bill を Tom と Sue に紹介した場合，(1a) は真になるが，(1b) は偽になる．一方，Mary が Bill と Tom を Sue に紹介した場合，真偽値は逆になり，(1a) が偽，(1b) が真になる．直感的に，*only* は「その命題を満たすのはフォーカスされたもののみである」という意味をもつ．つまり，*only* はフォーカスされた言語表現を抽象化した命題を要求し，抽象化された部分にフォーカスである言語表現を代入した命題は真，他の言語表現を代入したものは偽としている．この抽象化した命題は，(1a) では “Mary introduced x to Sue”，(1b) では “Mary introduced Bill to x” というものになる．これらはフォーカスに対するコンテキスト，すなわち「継続」である．

(1) のフォーカスは排他的な意味をもつ *only* に対するものだったが，このような副詞を伴わない場合においても，フォーカスは対比のニュアンスをもつ。以下の質問と応答がその例である．

(2) A: Who did Mary introduce to Sue?
B: Mary introduced $[\text{Bill}]_{\mathrm{F}}$ to Sue.

(3) A: To whom did Mary introduce Bill?
B: Mary introduced Bill to $[\text{Sue}]_{\mathrm{F}}$.

(2B) の応答は，Mary が他の誰でもなく Bill を Sue に紹介したことを意味する．一方 (3B) は，Mary が Bill を紹介した相手が他の誰でもなく Sue であることを意味する．両者とも “Mary introduced Bill to Sue” という命題が真になる状況であるが，この命題のみからフォーカスの位置による意味の違いを説明することはできない．フォーカスを含む文を解釈するためには，フォーカスの alternatives，すなわち比較対象を考慮する必要がある．このような考えを出発点としているのが alternative semantics (Rooth [16]) である．

Montague [14] 以来の形式意味論では，文や言語表現の意味は semantic value としてとらえられる．名前に対する semantic value はその名前が指し示すもの，動詞に対してはその属性を満たす個体の集合とする．文の semantic value は各構成要素の semantic value とそれらの結合の仕方によって定まる真理値である．ここで，文 α の semantic value を $[\![\alpha]\!]^o$ と表記することにする．Rooth [17, 18] によると，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した focus semantic value $[\![\alpha]\!]^f$ をもつ．Focus semantic value はフォーカスによって引き起こされる alternative の集合であり，フォーカスに対してはその比較対象の集合，フォーカスを含む文に対してはフォーカスの部分に同じ型をもつ他の表現を代入した命題の集合となる．α を文 (2B) とすると，$[\![\alpha]\!]^o$ はフォーカスの影響を無視した “Mary introduced Bill to Sue” という命題，$[\![\alpha]\!]^f$ はフォーカスとなっている Bill の部分が抽象化され，そこに Bill の比較対象となる人物が代入された命題を集めたものとなる．Focus semantic value は，以下の定義に従って再帰的・構成的に得られる (Rooth [18])[2].

a. フォーカスが型 τ をもつ言語表現の場合，その focus semantic value は τ 型の 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 $\Phi(\alpha_1, \ldots, \alpha_k)$ という形の複合的な言語表現であるとき，α の focus semantic value は $\{\phi(x_1, \ldots, x_k) \mid \phi \in [\![\Phi]\!]^f \wedge x_1 \in [\![\alpha_1]\!]^f \wedge \ldots \wedge x_k \in [\![\alpha_k]\!]^f\}$ なる集合である．

この定義に従うと，(2B) の focus semantic value は以下のように計算される．ただし，m, s, b は Mary, Sue, Bill の denotation とする．また，$[_{\mathrm{VP}}\,]$ は [ ] 内が動詞句，$[_{\mathrm{S}}\,]$ は [ ] 内が文になることを意味する．

(2B) Mary introduced $[\mathrm{Bill}]_{\mathrm{F}}$ to Sue.

$$
\begin{aligned}
[\![[\mathrm{Bill}]_{\mathrm{F}}]\!]^f &= E\text{（個体の集合）} \\
[\![\mathrm{Mary}]\!]^f &= \{m\}\text{（}[\![\mathrm{Mary}]\!]^o\text{からなる単一集合）} \\
[\![\mathrm{Sue}]\!]^f &= \{s\}\text{（}[\![\mathrm{Sue}]\!]^o\text{からなる単一集合）} \\
[\![\mathrm{introduced}]\!]^f &= \{\mathit{introduce}\}\text{（}[\![\mathrm{introduced}]\!]^o\text{からなる単一集合）} \\
[\![[_{\mathrm{VP}}\,\mathrm{introduced}\ [\mathrm{Bill}]_{\mathrm{F}}\ \mathrm{to\ Sue}]]\!]^f &= \{\lambda x.\,\mathit{introduce}\,(x, y, s) \mid y \in E\} \\
&\quad\text{（“introducing } y \text{ to Sue”という形の属性の集合）} \\
[\![[_{\mathrm{S}}\,\mathrm{Mary\ introduced}\ [\mathrm{Bill}]_{\mathrm{F}}\ \mathrm{to\ Sue}]]\!]^f &= \{\mathit{introduce}\,(m, y, s) \mid y \in E\} \\
&\quad\text{（“Mary introducing } y \text{ to Sue”という形の命題の集合）}
\end{aligned}
$$

本稿では，最終的に得られる命題の集合を alternative set とよぶ．

一般に，フォーカスの比較対象として考慮されるのは，フォーカスと同じ型を持つもの全てではない．(2B) において，Bill と比較されるのは全ての人間ではなく，(2B) が発話された文脈のなかで

[2]継続を用いた場合，文の focus semantic value を各構成要素の focus semantic value から再帰的に計算する必要がない．

関連のある人物のみに限られる．たとえば，Mary が Bill を Sue に紹介したときに Tom と Alice がその場にいた場合，Bill の比較対象は Tom と Alice であり，(2B) に対する alternative set は {Mary introduced Bill to Sue, Mary introduced Tom to Sue, Mary introduced Alice to Sue} という集合になる．

次に，フォーカスを伴う副詞 *only* の意味を考える．(1a) は (2B) と *only* を組み合わせたものであるが，これに対して以下のような表示を与えることができる．

(4) $[_{\mathrm{S}}\ \mathrm{only}(C)\ [_{\mathrm{S}}\ \mathrm{Mary\ introduced\ [Bill]_F\ to\ Sue}]]$

この表示では，*only* を命題に対する量化子としてとらえている．C は文脈によって制限された alternative set であり，“Mary introduced Bill to Sue” と，Bill の部分が他の人物に置き換えられた命題を少なくとも一つ含む．この集合が *only* の量化のドメインとなっている．Rooth [18] による *only* の意味表示は以下の通りである．なお，$^{\vee}q$ は与えられた可能世界と時点において命題 q を評価した際に，q が成り立つことを表す．

(5) $[\![\mathrm{only}]\!] = \lambda C.\, \lambda p.\, \forall q\, [((q \in C) \wedge {}^{\vee}q) \leftrightarrow (q = p)]$

p はフォーカスの部分にフォーカスされた言語表現が入った命題，q はフォーカス部分にフォーカスあるいは他の言語表現が入った命題である．C は p および p と異なる要素を一つ以上含む alternative set である．(5) は alternative set の要素 q が成り立つのは，q が p であるとき，またそのときのみであることを表している．言い換えると，フォーカス部分に何かを代入して成り立つのはそれがフォーカスされた言語表現であるとき，またそのときのみであることを意味する．(1a) の場合，(5) における p は “Mary introduced Bill to Sue” という命題であり，Bill の部分にある人物を代入した命題 $q\,(\in C)$ が成り立つのは，その人物が Bill であるとき，かつそのときのみとされる．

3 限定継続

3.1 継続とは

プログラミングにおける「継続」とは，ある時点における残りの計算のことを指す．たとえば，$1 + (2 * 3)$ の $2 * 3$ の部分を計算しているときの継続は,「現在行なっている計算の結果が返ってきたら，それに 1 を足す」という計算である．これは $\lambda x.\,(1 + x)$ という関数で表すことができる．

その後の計算全体ではなく，一部のみを取り出したいときは，限定継続を用いる．限定継続とは，範囲の限られた継続のことである．限定継続を扱うための命令はいくつかあるが，本稿では `shift/reset` (Danvy and Filinski [5]) および `fcontrol/run` (Sitaram [21]) を取り上げる．

3.2 `shift/reset`

本研究では，限定継続命令として `shift/reset` を採用する．`shift` はその命令が実行された時点の継続を関数として切り取る命令，`reset` は `shift` が切り取る継続の範囲を区切る命令である．例として以下の計算を考える．

(6)
```
1 + reset (2 * shift k. (3 + k 4))
```

この場合，`shift` によって捕捉される継続 `k` は，`reset` に囲まれた計算で，`shift` 節の部分が抽象化されたもの，つまり $\lambda x.\, \mathit{reset}\,(2 * x)$ という関数となる[3]．(6) を簡約すると，全体として 12 になる．なお，`shift` 節内に `k` は任意回現れることが可能である．`1 + reset (2 * (shift k. k (k 3)))` は 13 になり，`1 + reset (2 * (shift k. 3))` は 4 となる．

[3] ここで新たに `reset` が設けられるのが `shift/reset` の特徴である．詳しくは 3.3 節で述べる．

プログラムを直接形式 (direct style) で書いた場合，各部分項に対する継続は明示されない．継続を取り出してそれを計算の中で使うためには，プログラムを継続渡し形式 (Continuation-Passing Style, CPS) に変換する必要がある．以下に Plotkin [15] の CPS 変換規則（値呼び戦略）を示す．

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

継続渡し形式のプログラムでは，各関数に継続のための引数が追加され，それまでの計算結果が継続に渡される．

上述した `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．

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

`shift` $c.\, M$ は，その時点の継続 k を c に束縛し，式 M を恒等関数，すなわち空の継続で実行する．`reset` (M) は，式 M を空の継続で実行した結果を `reset` が呼び出された時点の継続に渡す．`shift/reset` オペレータを用いると，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，プログラムを継続渡し形式で解釈するインタプリタを用意すれば，計算を正しく評価することができる．

3.3 `fcontrol/run`

4.1 節で取り上げる Barker [2] の分析は，限定継続命令 `fcontrol/run` を用いている．`run` は継続を限定するための命令であり，プログラムおよびハンドラとよばれる手続きを受け取る．`run` は 2 引数の special form であり，第 1 引数の継続を限定して実行する．その際，中で `fcontrol` が呼ばれると，制御が `run` の第 2 引数に移る．`fcontrol` は `run` のプログラムに現れうるオペレータであり，引数を一つ要求する．この引数は，最も近い `run` までの継続とともにハンドラに渡される．例として `(+ 1 (run (* 2 (+ (fcontrol 3) 4)) (lambda (x k) (k (k x)))))` という計算を考えると，`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[4]．

`shift/reset` との主な違いは二つある．まず，`shift/reset` では継続を使った計算が `shift` 節に書かれるのに対し，`fcontrol/run` では継続を限定する `run` の中に書かれる．次に，前者は静的な振る舞いを見せるが，後者は動的な振る舞いを見せる．この性質の違いは，以下のプログラムに現れている．ここでは対応をとりやすくするために限定継続命令 `control/prompt` (Felleisen [7]) を用いているが，これは `fcontrol/run` と非常に類似した命令である (Shan [20])．

(7) a.
```
(reset (let ((y (shift f (cons 1 (f empty))))) (shift g y)))
↝ (reset (cons 1 ((lambda (x) (reset (let ((y x)) (shift g y)))) empty)))
↝ (reset (cons 1 (reset (let ((y empty)) (shift g y)))))
↝ (reset (cons 1 (reset (shift g empty))))
↝ (reset (cons 1 (reset empty)))
↝ (1)
```

[4] この例は Racket のプログラムである．Racket の control パッケージを読み込むと，`fcontrol/run` や `control/prompt`, 6.4 節の `shift0/reset0` といった限定継続命令を使用することができる．

b. `(prompt (let ((y (control f (cons 1 (f empty))))) (control g y)))`
$\rightsquigarrow$ `(prompt (cons 1 ((lambda (x) (let ((y x)) (control g y))) empty)))`
$\rightsquigarrow$ `(prompt (cons 1 (let ((y empty)) (control g y))))`
$\rightsquigarrow$ `(prompt (cons 1 (control g empty)))`
$\rightsquigarrow$ `(prompt empty)`
$\rightsquigarrow$ `()`

(7a) において，一つ目の `shift` で切り取られる継続 `f` は $\lambda x.(reset\ (let\ ((y\ x))\ (shift\ g\ y)))$ という計算となる．`shift` によって捕捉される継続に対し，それを限定する `reset` が新たに設けられるのが `shift/reset` の特徴である．これによって，二つ目の `shift` が 1 を加えるという計算を捕捉することが妨げられ，継続 `g` が破棄されても 1 は加えられる．一方，(7b) では捕捉された継続に `prompt` が挿入されないため，`g` は $\lambda x.(cons\ 1\ x)$ という計算になる．この `g` は破棄されるため，`empty` が結果となる．二つ目の `control` の切り取る継続が一つ目の `control` 節の計算に左右されるため，動的であるといわれる．`fcontrol/run` も `control/prompt` と同様に，`reset` のような限定子を含まない継続を切り取る．

4 限定継続を用いたフォーカスの分析

第 1 節で述べたように，自然言語で記述された文のなかで，ある言語表現の周りのコンテキストは，プログラミングにおける継続とみなすことができる．継続を使うと，Rooth [18] のような再帰的な計算をせずに，フォーカスの周りのコンテキストを取り出すことが可能になる．本節では，限定継続を用いてフォーカスの意味表示を与えている二つの先行研究について概説する．

ただし，先行研究が与えている意味表示は実行可能なプログラムになっていないため，イタリック体で記述してある．本研究において OCaml や OchaCaml, あるいは Racket で実行したものについてはタイプライタ体で記述した．

4.1 `fcontrol/run` を用いた分析

Barker [2] は `fcontrol/run` を用いてフォーカスの意味表示を与えている．*only* とそのフォーカス M は，Racket のスタイルで以下のように定義される．

$$[\![[M]_{\mathrm{F}}]\!] \stackrel{\mathrm{def}}{\equiv} fcontrol\,[\![M]\!]$$
$$[\![only\ P]\!] \stackrel{\mathrm{def}}{\equiv} runP\lambda x.\lambda k.(and\ (k\ x)\ (\forall z\,(or\ (equal\ x\ z)\,(not\ (k\ z)))))$$

`fcontrol` にフォーカスとなっている言語表現を渡すと，その言語表現と周りのコンテキストが `run` に渡される．`run` のハンドラはこれらを用いてフォーカスの意味を計算している．ハンドラに書かれている式は，フォーカス部分にフォーカスされた言語表現 x を代入した命題（(5) における p）が真であり，かつ全ての z について，フォーカス部分に z を代入した命題が成り立つなら，z は x であることを意味する．

上の定義に基づいて “John only drinks $[\text{Perrier}]_{\mathrm{F}}$” の意味表示を与えると以下のようになる．

(8) John only drinks $[\text{Perrier}]_{\mathrm{F}}$.
$$\begin{aligned}&[\![(run\ (drinks\ (fcontrol\ Perrier)\ j)\\ &\qquad (lambda\ (x\ k)\ (and\ (k\ x)\\ &\qquad\qquad (\forall z\ (or\ (equal\ x\ z)\ (not\ (k\ z))))))]\!]\\ &= (and\ (drinks\ Perrier\ j)\\ &\qquad (\forall z\ (or\ (equal\ Perrier\ z)\ (not\ (drinks\ z\ j)))))\end{aligned}$$

(8) において，*drinks* の第 1 引数に目的語 (*fcontrol Perrier*) が渡されることによって，継続 k は *Perrier* の部分が抽象化された命題，すなわち $\lambda x.(drinks\ x\ j)$ という関数になる．この k の抽象化された部分にさまざまな言語表現を代入した命題を集めると alternative set が得られる．

4.2 `shift/reset` を用いた分析

Bekki and Asai [3] は，`shift/reset` オペレータを使ってフォーカスの分析を行っている．*only* とそのフォーカスに対する定義は以下の通りである．

$$\begin{aligned} [M]_{\mathrm{F}} &\stackrel{\mathrm{def}}{\equiv} shift\, k.\forall x\,(k\,x \leftrightarrow x = M) \\ only\,(\phi) &\stackrel{\mathrm{def}}{\equiv} reset\,(\phi) \end{aligned}$$

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．

この定義に従って (1a), (1b) の意味表示を与えると，以下のように簡約される．

(1) a. Mary only introduced $[\text{Bill}]_{\mathrm{F}}$ to Sue.

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

b. Mary only introduced Bill to $[\text{Sue}]_{\mathrm{F}}$.

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

4.3 両者の比較

Barker [2] はフォーカスを `fcontrol` で統一的に扱い，`run` のハンドラの中で副詞の意味を記述していた．一方，Bekki and Asai [3] は副詞を単なる `reset` とし，`shift` でそれぞれの副詞に特化したフォーカスを表現している．副詞に具体的な意味を与えない後者のアプローチは，幾分副詞を一般化しすぎているように思えるが，一方で (2), (3) に示した例のように，フォーカスには副詞を伴わない用法も存在する．Barker の定義に従うとこのような文は表現することができないが，Bekki らの定義では，副詞のない場合にその文全体を `reset` で囲むようにすれば可能である．

また，`shift/reset` と `fcontrol/run` の振る舞いの違いによって，同じ文に対する意味表示の簡約結果が異なる場合がある．ここで，副詞 *also* が「その命題を満たすもので，フォーカス以外のものが存在する」という前提をもつとする．この前提は $shift\,k.\exists x\,(k\,x \wedge \neg(x = M))$ と表すことができる[5]．これをふまえて以下の文を考える．

(9) Mary also only introduced $[\text{Bill}]_{\mathrm{F_o}}$ to $[\text{Sue}]_{\mathrm{F_a}}$.

(9) は「Sue 以外に Mary が Bill だけを紹介した人が存在する」という意味をもつ．$[\ ]_{\mathrm{F_o}}$ と $[\ ]_{\mathrm{F_a}}$ はそれぞれ *only* と *also* のフォーカスを表す．この文に対する意味表示を `fcontrol/run` および `shift/reset` で記述すると，それぞれ以下のように簡約される（ただし，*bicond* は $\leftrightarrow$ を表す）．

[5] *also* に対するフォーカスの意味表示は，$[M]_{\mathrm{Fa}} \stackrel{\mathrm{def}}{\equiv} shift\,k.\,(\exists x\,(k\,x \wedge \neg(x = M)) \wedge k\,M)$ と定義できる．$k\,M$ はフォーカスされた言語表現がその命題を満たすことを表すが，ここでは前提のみを考慮する．

$$
\begin{aligned}
&(run\,(run\,(introduce\ (fcontrol\ b)\ (fcontrol\ s)\ m)\\
&\qquad (lambda\ (v\ k)\ (\forall x\,(bicond\ (k\ x)\ (x = v)))))\\
&\quad (lambda\ (v\ k)\ (\exists y\,(and\ (k\ y)\ (not\ (y = v))))))\\
&= \exists y\,(\forall x\,(introduce\,(m,\,x,\,y) \leftrightarrow (x = b)) \wedge \neg(y = m))
\end{aligned}
$$

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

`fcontrol/run` では，二つ目の `fcontrol` が切り取る継続に内側の `run` のハンドラの計算が含まれているため，$\exists$ が外側にかかる表示に簡約される．一方，`shift/reset` を用いた場合，一つ目の `shift` で切り取られる継続が新たに設けられた `reset` で囲まれることによって，二つ目の `shift` が切り取る継続の範囲は一つ目の `shift` の継続までに限られる．そのため，$\exists$ が $\forall$ の内側におさまる表示となる．この例文では $\exists$ のスコープが上となる表示が好ましいが，第 5 節のインタプリタで関数適用に対する CPS 変換を $[\![M\,N]\!] = \lambda k.\,[\![N]\!](\lambda n.[\![M]\!](\lambda m.\,(m\,n)\,k))$ として右から実行するようにすれば，`shift/reset` でも期待された表示に簡約される．本研究では `shift/reset` を採用する．

5　インタプリタの実装

5.1　CPS インタプリタ

叢ら [22] は，Bekki and Asai [3] の定義に従った意味表示を簡約するインタプリタを関数型言語 OCaml で実装した[6]．

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*, *think*，三項述語 *introduce* を追加したものである（図 1）．*love* と *introduce* はそれぞれ関数として定義してある．たとえば *love* は目的語 y と主語 x を受け取ったらコンストラクタ $love\,(x,\,y)$ を返す．

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

図 1. 対象言語

入力 → 字句解析 → 構文解析 → CPS 変換 → 評価 → 出力

$$
reset\,(love\,(shift\,k.\,forall\,x\,(k\,x \leftrightarrow x = m))\ j) \quad \Longrightarrow \quad \forall x\,(love\,(j,\,x) \leftrightarrow x = m)
$$

図 2. 処理の流れ

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，空の継続のもとで簡約される（図 2）．`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．

[6] 将来的には，自然言語のテキストを入力とし，それを意味表示に変換して解釈することを目指しているが，現在は入力時にフォーカスの位置を指定し，それを `shift/reset` で表現した意味表示を与えている．

図 3 に実行例を示す．1, 2 は第 2 節の例文 (1a), (1b) に対する意味表示，3 はフォーカスが動詞にある場合，4 は *only* を伴わない場合である．

```
(* 1. Mary only introduced [Bill]F to Sue. *)

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


(* 2. Mary only introduced Bill to [Sue]F. *)

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


(* 3. Mary only [introduced]F Bill to Sue. *)

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


(* 4. Mary introduced [Bill]F to Sue. *)

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

図 3. 実行例

フォーカスが動詞にある場合は，名詞にある場合と異なり，特殊な扱いをする必要がある．1 に示した例において，`shift` によって切り取られる継続 `k` は $\lambda y.\, reset\,(introduce\,(m,\, y,\, s))$ という関数であり，これが全称量化子の束縛する変数 `x` に適用される．一方，3 において，ユーザ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意図している．この場合，継続 `k` は `p` に対する関数適用であり，`k p` は `p b s m` 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「`p` は関数でないので，これを適用することはできない」というエラーが生じてしまう．これを回避するために，関数適用の一つ目の要素が単なる変数だった場合は，ユーザの入力した意味表示が全うであるという仮定のもとで，それを関数として解釈するようにした．

また，`shift` 命令で切り取られる継続は `reset` によって限定されることが前提とされている．そのため，6 節で扱う OchaCaml では，4 のように `reset` を伴わずに `shift` 命令を実行することが許されない．CPS インタプリタでこのような計算が可能になるのは，初期継続として恒等関数を渡しており，それが実質的に `reset` と同じ役割を果たしているためである．

5.2 問題点

叢ら [22] のインタプリタを使うと，一つの言語表現が複数の副詞にフォーカスされている場合に，簡約結果が好ましくないものになってしまうことがある．Krifka [9] に (10) のような例がある．

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

2 文目は，ワイン以外のもので，John がそれだけを飲んでいたものが存在することを意味する．“wine” がネストしたフォーカスになっており，外側は *only*, 内側は *also* に対応している．また，*also* と *only* のスコープ関係は前者が上となっている．このネストしたフォーカスは以下のように構成される．

(11) also $[_{\mathrm{S}}\ [\mathrm{wine}]_{\mathrm{F_a}}\ \lambda e_2\ [_{\mathrm{S}}$ have $[_{\mathrm{S}}$ only $[_{\mathrm{S}}\ [e_2]_{\mathrm{F_o}}\ \lambda e_1\ [_{\mathrm{S}}$ He drunk $e_1]]]]]$

(11) において，“wine” はまず外側のスコープをとる *also* と直接結びついて *also* のフォーカス $[\mathrm{wine}]_{\mathrm{F_a}}$ となり，その値が継続渡し形式のプログラムのように e_2 として λ に束縛される．内側のスコープをとる *only* はこの e_2 をフォーカス対象として受け取り，結果として $[[\mathrm{wine}]_{\mathrm{F_a}}]_{\mathrm{F_o}}$ という形のネストしたフォーカスが構成される．

(10) の 2 文目の意味は以下のように表されるのが適切である．

(12) $\exists y\,(\forall x\,(\mathit{drink}\,(j,\,x) \leftrightarrow (x = y)) \wedge \neg(y = b))$

Bekki and Asai [3] の手法に従うと，ネストしたフォーカスは `shift` 命令を 2 回実行することで表現できる．つまり，以下のような表示となる（$\mathit{shift}_o/\mathit{reset}_o$, $\mathit{shift}_a/\mathit{reset}_a$ はそれぞれ *only* と *also* に対応している）．

(13) $\mathit{reset}_a\,(\mathit{reset}_o\,(\mathit{drink}\,(j,\,\mathit{shift}_o\,k_o.\forall x\,(k\ x \leftrightarrow x = \mathit{shift}_a\,k_a.\,\exists y\,(k\ y \wedge \neg(y = b))))))$

この中で，*also* に対する $\mathit{shift}_{\mathrm{a}}$ は *only* に対する $\mathit{reset}_{\mathrm{o}}$ を越えて，$\mathit{reset}_{\mathrm{a}}$ までの継続を取ってくる必要がある．しかし，通常の `shift/reset` オペレータで切り取られるのは，最も内側の `reset` までの継続である．この場合，*only* と *also* のフォーカスに対するコンテキストの範囲が同一であり，二つの `reset` の間に計算が含まれないため，(13) は (12) に簡約されるが，(14) のような例では簡約結果が好ましくないものとなってしまう．

(14) Sue has also thought that John only loves $[[\mathrm{Mary}]_{\mathrm{F_a}}]_{\mathrm{F_o}}$.
$\mathit{reset}_a\,(\mathit{think}\,(s,\,\mathit{reset}_o\,(\mathit{love}\,(j,\,\mathit{shift}_o\,k_o.\forall x\,(k\ x \leftrightarrow x = \mathit{shift}_a\,k_a.\,\exists y\,(k\ y \wedge \neg(y = m))))))$

(14) は「Mary でない人で，John が彼女だけを愛していると Sue がかつて思っていた人が存在する」という読みをもつ．この文においては，*only* と *also* のフォーカスに対するコンテキストが異なっている．前者のコンテキストは「John が誰かを愛している」という部分に限られるが，後者のコンテキストは Sue の信念も含んでおり，「Sue は John が誰かだけを愛していると思っていた」というものになる．これを表現しているのが (15) である．

(15) $\exists y\,(\mathit{think}\,(s,\ \forall x\,(\mathit{love}\,(j,\,x) \leftrightarrow (x = y)) \wedge \neg(y = m)))$

(15) において，*only* のスコープは “John loves x”，*also* は “Sue has thought that John only loves x” となっている．(14) の意味表示をこの形に簡約するためには，$\mathit{shift}_{\mathrm{a}}$ で think までの継続を切り取る必要がある．しかし，通常の `shift/reset` を用いると，(14) は (16) のように簡約される．

(16) $\mathit{think}\,(s,\ \exists y\,(\forall x\,(\mathit{love}\,(j,\,x) \leftrightarrow (x = y)) \wedge \neg(y = m)))$

also の意味表示に含まれる存在量化子は，本来 think の外側にかかるべきであるが，(16) では存在量化子が *only* の全称量化子と同じ love までのスコープを取る表示となっている．それぞれの `shift` で対応する `reset` までの継続を切り取るには，*only* と *also* のフォーカスに対する継続を区別する必要がある．これを実現するためには，たとえば CPS 変換を二回行い，二つの異なる階層の継続を扱うという手法が考えられる．

6 OchaCaml を用いた実装

前節で述べた問題を解決するために，本研究では OchaCaml を使って二つの階層の限定継続命令を定義し，それらを用いてフォーカスの分析の実装を行った．

6.1 OchaCaml

Caml 言語の一つである Caml Light は，軽量かつ移植性の高い言語である (Leroy [10])．OCaml と比べて表現力やオブジェクト指向性などの面で劣るが，その分型推論が幾分容易である．Masuko and Asai [11] はこのメリットを生かして，Caml Light に `shift/reset` を直接実装している．この言語を OchaCaml とよぶ．

OchaCaml における `shift/reset` のシンタクスおよび実行例を以下に示す．

```
shift (fun <var> -> <exp>)
reset (fun () -> <exp>)

# 1 + (reset (fun () -> 2 * shift (fun k -> 3 + k 4))) ;;
- : int = 12
```

6.2 shift2/reset2

5.2 節において，(14) のような文の意味表示を正しく簡約するためには，二つの異なる階層の継続が必要であると述べた．Danvy and Filinski [5] は，CPS 変換を任意回行うことで，任意個の階層化限定継続を定義できることを示しているが，本研究では，OchaCaml を用いることで CPS を介さずに `shift/reset` より一つ階層が上の `shift2/reset2` を定義した．

アプローチとしては，以下の二つのアイディアをもとにしている．

- 全ての monadic effect は `shift/reset` を使うと直接形式で表現することができる (Filinski [8])
- `shift2/reset2` のような階層化限定継続も monadic effect の一つとして書くことができる

ここでは，通常の `shift/reset` にあたるオペレータを `shift1/reset1` とする．`shift2` は `reset1` を越えて `reset2` までの継続を切り取ることができる．二つのオペレータを定義する際に，OchaCaml に備わっている `shift/reset` を使用している．以下に `shift1/reset1`, `shift2/reset2` を使ったプログラムの例を示す．実装の都合上，式全体を `run1 (fun () -> )` で囲む必要があるが，この操作は CPS インタプリタでプログラムを評価する際に，初期継続として恒等関数を与えているのと対応している．

```
# run1 (fun () ->
       reset2 (fun () -> 1 + reset1 (fun () -> 2 * shift2 (fun k -> k (k 3))))) ;;
- : int = 15
```

この例を通常の `shift/reset` のみで計算すると，`k` に束縛される継続は $\lambda x.\, reset\,(2 * x)$ となり，全体として 13 になる．一方，`shift2` を使うと，`reset2` までの継続，すなわち $\lambda x.\, reset2\,(1 + (2 * x))$ を切り取ることができる．よって，`k (k 3)` は `1 + (2 * (1 + (2 * 3))` となり，結果は 15 になる．

6.3 OchaCaml を用いたフォーカスの意味計算

本研究では，OchaCaml 上で `shift1/reset1` および `shift2/reset2` を用いてフォーカスの意味表示を与えた．

意味表示に必要な論理演算や述語は，ここでは全て文字列をつなげる関数にしている．たとえば，述語 *love* は引数 y, x を受け取ったら，文字列 "$love\,(x,\, y)$" を返す．これらを用いて 5.2 節の (14) の意味表示を与えると，以下のように簡約される（`bicond`, `eq`, `conj`, `neg` はそれぞれ $\leftrightarrow$, $=$, $\wedge$, $\neg$ を表す）．

(14) Sue has also thought that John only loves $[[\mathrm{Mary}]_{\mathrm{F_a}}]_{\mathrm{F_o}}$.

```
# run1 (fun () ->
    reset2 (fun () -> think
      reset1 (fun () -> love
        shift1 (fun k1 -> forall "x" (bicond (k1 "x") (eq "x"
          shift2 (fun k2 -> exists "y" (conj (k2 "y") (neg (eq "y" m)))))) j) s)) ;;
- : string =
 "exists y (think s, (forall x (love (j, x) <-> x = y)) and not (y = m))"
```

通常の shift/reset を用いた場合と異なり，*also* の存在量化子が think の外側にかかっている．これは (14) の意味を正しく表現している．よって，二つの副詞によってフォーカスされた言語表現を含む文においては，それぞれのフォーカスに対応するコンテキストを階層付けした二つの限定継続命令で区別できることが示された．

6.4 さらなる拡張の必要性

shift2/reset2 を定義したことにより，二つの階層の限定継続を扱うことが可能になったが，さらなる階層化は必要なのだろうか．経験的に，一つの文に含まれるフォーカスは高々二つまでであり，三つ以上のケースは日常会話においてみられないように思える．人工的につくられた「不自然な」文も説明対象とする，という選択肢もあるが，本研究ではフォーカスが二つまでの場合を考慮するだけで十分であるという立場を取る．

ただ，一つ興味深いのは，さまざまな限定継続命令を使って (14) の意味表示を記述したところ，shift0/reset0 (Danvy and Filinski [4]) を用いると，*only* と *also* の継続を明示的に区別しなくても，正しい表示に簡約されたことだ．

shift0 は，呼び出す際に最も内側の reset0 を除去するという特徴をもつ．ネストした二つの shift0 が二つの reset0 に囲まれている計算において，外側の shift0 は内側の reset0 までの継続を切り取り，内側の reset0 をキャンセルする．そのため，内側の shift0 が切り取る継続は外側の reset0 までの計算となる．以下の計算は，shift/reset と shift0/reset0 の差が出る例である．

(17) a.
```
(reset (cons 1 (reset (shift f (shift g empty)))))
↝(reset (cons 1 (reset (shift g empty))))
↝(reset (cons 1 empty))
↝(1)
```

b.
```
(reset0 (cons 1 (reset0 (shift0 f (shift0 g empty)))))
↝(reset0 (cons 1 (shift0 g empty)))
↝(reset0 empty)
↝()
```

この性質は，ネストしたフォーカスを含む文の意味表示を簡約する際に，内側の shift0 で外側の reset0 までの継続が切り取られることを意味する．実際に，Racket で shift0/reset0 を用いて (14) の意味表示を記述すると，以下のように簡約される．

```
> (reset0 (think
        reset0 (love
             shift0 k (forall "x" (bicond (k "x") (eq "x"
                  shift0 k1 (exists "y" (conj (k1 "y") (neg (eq "y" "m")))))) j) s))
"exists y (think (s, forall x (love (j, x) <-> x = y)) and not (y = m))"
```

shift0/reset0 を使えば，それぞれの副詞に対する継続の階層を指定する必要がなくなり，さらに任意回ネストしたフォーカスを表現することが可能になる．そのかわり，shift0/reset0 では

副詞のスコープ関係にかかわらず，フォーカスのネストの仕方によって簡約のされ方が決定してしまう．つまり，意味表示が期待通りに簡約されるのは，n 個の副詞とそれぞれに対応するフォーカスが $adv_1\, adv_2 \ldots adv_n \ldots [\ldots[[M]_{\mathrm{F}_1}]_{\mathrm{F}_2}\ldots]_{\mathrm{F}_n}$ という形で出現するというときに限られる．そのため，`shift0/reset0` を用いた分析が可能であるかを判断するには，英語におけるネストしたフォーカスが必ずこのような構造をもつか否かを検証する必要がある．3 回以上ネストしたフォーカスを含む文についての検証はまだ行っていないが，5.2 節の (11) のようなプロセスをたどれば，任意のネストしたフォーカスは上記の構造をもつことが予測される．そのため，ネストしたフォーカスをより一般的に扱えるオペレータとして `shift0/reset0` を採用することは一つの可能性として考えている．

7　おわりに

本研究では，限定継続命令を用いてフォーカスの意味表示を与えた．ネストしたフォーカスを含む文において，それぞれの副詞のフォーカスに対応するコンテキストを区別するために，OchaCaml で `shift1/reset1` および `shift2/reset2` を定義した．実際に，これらを用いてネストしたフォーカスを含む文の意味表示を与え，それが正しく簡約されることを確認した．

冒頭で述べたように，逆スコープを含む文も，意味を考慮する際にコンテキストが必要となる．継続を用いた逆スコープの分析としては Barker [2], Bekki and Asai [3] があるが，いずれも実装は行っていない．今後は，この現象に対して `shift/reset` による定式化および実装を行う予定である．さらに，フォーカスと逆スコープが混在する文における両者の相互作用についても考察したいと考えている．

謝辞 多くの有益なコメントを下さった査読者の皆様に感謝いたします．

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] 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).

[4] Danvy, O. and Filinski, A.: A Functional Abstraction of Typed Contexts, Technical Report 89/12, DIKU, University of Copenhagen (1989).

[5] Danvy, O. and Filinski, A.: Abstracting Control, In: *LFP90, the 1990 ACM Conference on Lisp and Functional Programming*, pp. 151–160 (1990).

[6] de Groote, P.: Type raising, continuations, and classical logic, In: R. van Rooij and M. Stokhof (eds.): *the 13th Amsterdam Colloquium*, Institute for Logic, Language and Computation, Universiteit van Amsterdam, pp. 97–101 (2001).

[7] Felleisen, M.: The Theory and Practice of First-Class Prompts, *Conference Record of the 15th Annual ACM Symposium on Principles of Programming Languages*, pp. 180–190 (1988).

[8] Filinski, A.: Representing Monads, In: *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Portland, Oregon, pp.446-457 (1994).

[9] 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).

[10] Leroy, X.: *The Caml Light system release 0.74* (1997).

[11] Masuko, M. and K. Asai .: Caml Light + shift/reset = Caml Shift, Theory and Practice of Delimited Continuations (TPDC 2011), pp. 33–46 (2011).

[12] May, R.: *The Grammar of Quantification*, Doctoral dissertation, MIT, Cambridge (1977).

[13] May, R.: *Logical Form: Its Structure and Derivation*, MIT Press, Cambridge (1985).

[14] Montague, R.: The Proper Treatment of Quantification in Ordinary English, In: J. Hintikka, J. Moravcsic, and P. Suppes (eds.): *Approaches to Natural Language*, Dordrecht, Reidel, pp. 221–242 (1973).

[15] Plotkin, G. D.: Call-by-Name, Call-by-Value and Lambda Calculus, *Theoretical Computer Science* 1, pp. 125–159 (1975).

[16] Rooth, M.: *Association with Focus*, PhD thesis, University of Massachusetts, Amherst, GLSA, Dept. of Linguistics, South College, UMASS, Amherst MA 01003.

[17] Rooth, M.: A Theory of Focus Interpretation, *Natural Language Semantics* 1, pp. 75–116 (1992).

[18] Rooth, M.: Focus, *The Handbook of Contemporary Semantic Theory*, pp.271–298 (1997).

[19] 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).

[20] Shan, C.-c: Shift to Control, In: Shivers, O. and Waddell, O. (eds.): *Proceedings of the 2004 ACM SIGPLAN Workshop on Scheme and Functional Programming*, Technical report TR600, Computer Science Department, Indiana University, Snowbird, Utah, pp. 99–107 (2004).

[21] Sitaram, D.: Handling Control, In: *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, ACM Press, pp. 147–155 (1993).

[22] 叢悠悠, 浅井健一, 戸次大介: 限定継続を用いたフォーカスの分析と実装に向けて, 情報処理学会第 214 回自然言語処理研究会 (2013).

[23] 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).