プログラミングー言語・基礎・実践ー　1－3
（1991. 4. 26）

継続つきプログラミング言語と線形論理

西崎真也
京都大学数理解析研究所

線形論理の枠組のもとで継続つきプログラムについて考察した。継続つきプログラミング言語から線形論理への変換を定義し、継続プリィミティブ *call/cc* が証明ネットで記述でき、線形論理が継続つきのプログラムを記述する能力があることを示した。継続つきプログラムの特徴として、計算結果が評価戦略に依存することがあるが、この変換は評価戦略に関する情報を付加する。

PROGRAMMING LANGUAGE WITH CONTINUATION AND LINEAR LOGIC

Shin-ya Nishizaki
Kyoto University Research Institute for Mathematical Sciences
Kitashirakawa-Oiwakecho Sakyo-ku,Kyoto 606 Japan

The program with continuation is studied under the framework of linear logic. The transformation from programming language with continuation to linear logic is defined, and we show that program with continuation can be described by linear logic. It is the most remarkable feature of program with continuation that the result of a program with continuation is dependent on a evaluation strategy. This transformation add the information of evaluation strategy to the program.

1 継続と関数型言語

継続 (continuation) は Scott-Strachey の表示的意味論 [Sto77] において、大域脱出の意味付けをするために生まれてきた概念である。継続とは、大まかにいえば、"残りの計算 (the rest of the computation)" である。継続に関する様々な研究が行なわれてきた。[Wan78] [FFKD86] [FFKD87]

継続という概念の応用の一つに、継続を自在に操作できるようにした Lisp の方言 Scheme がある。([RC86])

Scheme においては プリミティブ call-with-current-continuation(以下簡単のため、call/cc とよぶ) により継続を取り扱う。call/cc のもっとも簡単な使用例をあげる:

例 1 [call/cc による大域脱出の実現]

リスト (54 0 37 -3 245 19) の中から負の要素を求める。もし負の要素がないならば、#t を返す。

```
(call/cc
  (lambda (exit)
    (for-each
      (lambda (x)
        (if (negative? x) (exit x)))
        '(54 0 37 -3 245 19))
      #t))

→ -3
```

■

Scheme では、継続を値と同様に自由自在に取り扱う機構を提供することにより、大域脱出のみならず、バックトラッキング、コルーチンなどの様々な制御構造を実現することが可能となっている。[HFW84][Hay86]

2 継続と値との双対性

継続と値との双対性が Filinski [Fil89] により指摘されている。

この双対性の顕著な一例として次の例があげられる。

型 $A \to B$ の関数の直感的な説明として、

> 型 A の値が与えられると型 B の値を得ることができる。

ということができるが、

> 型 B の継続が与えられると型 A の継続を得ることができる。

という前者と双対的な説明もできる。

3 継続つきのプログラミング言語と評価戦略

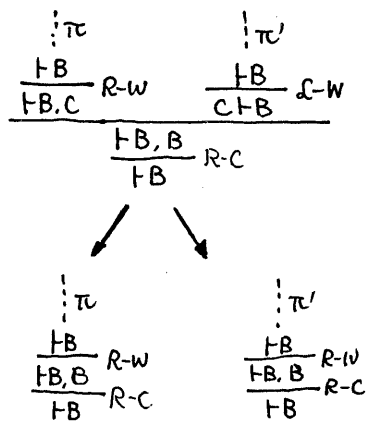
λ 計算は、Church-Rosser 性により計算結果が評価戦略に依存しないことが保証されているが、継続つきのプログラミング言語には、計算結果が評価戦略に依存するという特徴がある。例えば、次の例がもっとも顕著である:

```
(call/cc
    (lambda (exit) ((lambda (x) 1)(exit 2))))
```

は、名前呼び評価戦略 (call-by-name) に従うと結果は、 1 となり、値呼び評価戦略 (call-by-value) に従うと結果は、 2 となる。

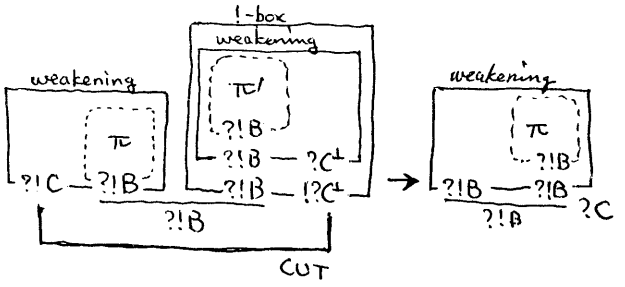
4 古典主義的論理と線形論理

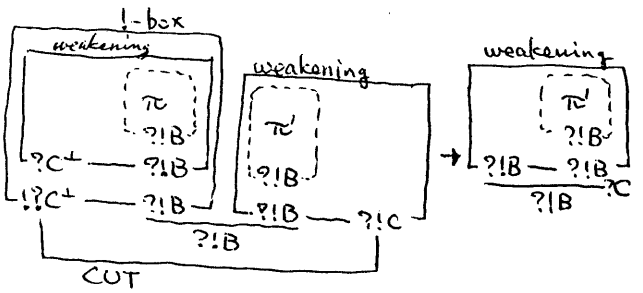
従来の古典主義的論理の特色の一つに、証明の正規化が非決定的であることがあげられる。[GTL89]



Jean-Yves Girard も指摘するように、この現象は、古典主義的論理では (sequent calculus において) 左辺と右辺の両方に weakening 規則 · contraction 規則が許されることに起因する。この現象を避けるために、直観主義的論理では、左辺の論理式の個数を高々一つ以下に制限して、それらの構造規則の右辺への適用を抑制することにより、正規化の非決定性を避けている。

これに対して、Girard により提唱された線形論理 (linear logic) は、weakening 規則 · contraction 規則をより厳密に扱うことにより、この種の非決定性を回避する。たとえば、上の古典主義的論理の証明に対応する線形論理の証明は次の二通りになる。





従って、古典主義的論理の証明を線形論理の証明に書きなおす際に決定性が付加されるとみなすこともできる。

5 古典主義的論理と継続つきプログラミング言語との対応

直観主義的一階命題論理と単純型理論との対応は、Curry-Howard の対応 (Curry-Howard isomorphism) として広く知られている。

Curry-Howard の対応は、Griffin [Gri90] により古典主義的命題論理にまで拡張された。彼により拡張された古典主義的論理は継続つきプログラミング言語に対応するのである。例えば、継続プリミティブ call/cc の型は、$((A \to B) \to A) \to A$ であるが、call/cc に対応する証明は、つぎのとおり。

$$\dfrac{\dfrac{\dfrac{\dfrac{A \vdash A}{A \vdash B, A}\,R\text{-}W}{\vdash A \to B, A}\,R\to \qquad A \vdash A}{(A \to B) \to A \vdash A, A}\,L\to}{(A \to B) \to A \vdash A}\,RC}{\vdash ((A \to B) \to A) \to A}\,R\to$$

$((A \to B) \to A) \to A$ を命題とみなすと、これは Pierce の法則とよばれる命題で、直観主義的 (命題) 論理では証明できないことで有名である。

6 本論文の動機

Griffin によりあたえられた、古典主義的論理と継続プリミティブを持つプログラミング言語との対応のもとでつぎの二つの現象が等しいことがわかる。

- 継続プリミティブを持つプログラミング言語で計算結果が評価戦略に依存するという現象
- 古典主義的論理における証明の正規化が非決定的になるという現象

継続を取り扱うことのできるプログラミング言語において、計算結果が評価戦略に依存しないようなものを考える枠組として線形論理を用いることは自然であろう。これが本論文の動機である。以下、本論文では、call/cc を付加した λ 計算から線形論理の証明 — 証明ネット — への変換を与え、プログラムとして証明ネットを考察してゆく。

7 体系 $\lambda \to_c$

本節では、単純型理論に定数としてプリミティブ call/cc を加えた体系を定義する。

定義 1 [型] 型 (type) は次のように定義される:

$$A ::= \alpha | A \to B$$

但し、$A, B, \cdots$ は型を、$\alpha, \cdots$ は原子型 (atomic type) をあらわす。$A \to B$ という形をした型を関数型 (function type) とよぶ。$\to$ は、右に結合的とする。すなわち、$A \to B \to C \equiv A \to (B \to C)$ である。 ∎

定義 2 [項] 項 (term) は次のように定義される:

$$M ::= x | \lambda x : A.\, M | (M\ N) | call/cc$$

但し、$M, N, \cdots$ は、項をあらわす。call/cc は定数である。x は変数をあらわす。 ∎

定義 3 [型環境] 型環境 (type environment) とは、変数と型との対の列のことであり、$\Gamma, \Delta, \cdots$ であらわす。例えば、

$$\Gamma \equiv [x : \alpha][y : \beta][f : \alpha \to \beta]$$

∎

定義 4　[型付けされた項] 項が型付けされている (typed) とは、次の型付け規則から導かれる項であることである。$\Gamma \vdash M : A$ が成り立つ時、項 M は、型環境 Γ のもとで型 A をもつという。

変数

$$\frac{[x : A] \in \Gamma}{\Gamma \vdash x : A}$$

但し、$[x : A] \in \Gamma$ とは、対 $[x : A]$ が列 Γ に含まれていることをあらわす。

λ 抽象

$$\frac{\Gamma[x : A] \vdash M : B}{\Gamma \vdash (\lambda x : A\,.\,M) : B}$$

関数適用

$$\frac{\Gamma \vdash M : (A \to B) \quad \Gamma \vdash N : A}{\Gamma \vdash (M\ N) : B}$$

定数 $call/cc$

$$\frac{}{\Gamma \vdash call/cc : ((A \to B) \to A) \to A}$$ ■

8　継続つきプログラムから線形論理の証明 — 証明ネット — への変換

本節では、継続つきプログラムの型から線形論理の命題への変換 τ と、継続つきプログラムから線形論理の証明 “証明ネット” への変換 T をあたえる。

8.1　線形論理による型付けの直感的な意味

変換を定義する前に、ラムダ計算から線形論理への従来の様々な変換に簡単にふれることにより、線形論理の論理記号 $\multimap, !, ?$ の計算機科学の立場からみた意味を考える。

もっとも簡単なものとして、(単純型つき) 線形 λ 計算 (Lafont [Laf88]) から線形論理への変換[1]がある。線形 λ 計算は、変数の出現がちょうど一回しか許されないような λ 計算である。例えば、$\lambda x.M$ を線形 λ 計算の λ 項とすると、変数 x の部分項 M での出現回数はちょうど 1 回でなければならない。この項に対する型づけをすると、その型は、線形論理の命題 $A' \multimap B'$ に対応する。(但し、A', B' は、A, B のおのおのに変換を (帰納的に) 施したもの。) したがって、命題 $A \multimap B$ は、

　型 A の項を受けとり、それを ちょうど一回使用して、型 B の項を返す関数の型

と考えることができる。

もう一つ、重要な変換として、System F から線形論理への変換 (Girard [Gir87]) がある。System F_c の型から線形論理の命題への変換 $(\cdot)^0$ は次の通り:

$$\begin{aligned} X^0 &= X \ \text{(但し、X は型変数とする)} \\ (A \Rightarrow B)^0 &= !A^0 \multimap B^0 \\ (\forall X.A)^0 &= \wedge X.A^0 \end{aligned}$$

$A \multimap B$ の説明に当てはめると、命題 $!A \multimap B$ は、「型 $!A$ の値をうけとり、それをちょうど一回使用して、型 B の項を返す型」ということになる。

型 $!A$ は、二重化[2]·消去[3]·読みだし[4]のできる型と考えることができる。

従って、$!A \multimap B$ は、

　型 A の項を受けとり、0 回以上使用して、型 B の項を返す型

と考えることができる。

再び、線形 λ 計算にもどる。型 $A \multimap B$ が型 A から型 B への関数の型であった。さて、$A \multimap B$ を線形論理の命題と見ると次の変換ができる。

$$\begin{aligned} A \multimap B &= A^\perp \wp B && \multimap \text{の定義} \\ &= B \wp A^\perp && \wp \text{の交換可能性} \\ &= (B^\perp)^\perp \wp A^\perp && \text{線形否定の定義} \\ &= B^\perp \multimap A^\perp && \multimap \text{の定義} \end{aligned}$$

型 A から型 B への関数は、型 $B^\perp$ から型 $A^\perp$ への関数ということになる。

Filinski[Fil89] の提唱した “値と継続の双対性” によると

　型 A の値から型 B へ値への関数
　= 型 B の継続から型 A への継続の関数

であった。

これらのことから次のことがわかる:

　型 $A^\perp$ の値 = 型 A の継続

まとめると次の通り:

　型 $A^\perp$ の値　=　型 A の継続
　型 A の値　=　型 $A^\perp$ の継続
　型 A の値から型 B の値への関数
　= 型 B の継続から型 A の継続への関数

[1] Lafont は線形論理への変換を与えていないが、型つけ規則をそのまま線形論理の推論規則に読みかえることできる。そうすることにより、λ 項の型付けの導出木はそのまま線形論理の証明に読みかえることができる。

[2] duplicate

[3] delete

[4] read

System F の関数の型は線形論理では $!A \multimap B$ であった:

$$\begin{array}{rcll} !A \multimap B & = & (!A)^{\perp} \wp B & \multimap \text{の定義} \\ & = & ?A^{\perp} \wp B & \text{線形否定の定義} \\ & = & B \wp ?A^{\perp} & \wp \text{の定義} \\ & = & B^{\perp\perp} \wp ?A^{\perp} & \text{線形否定の定義} \\ & = & B^{\perp} \multimap ?A^{\perp} & \multimap \text{の定義} \end{array}$$

このことから、(線形λ計算はもちろんのこと)System F であっても、継続の二重化・消去は許されないことがわかる。しかし、大域脱出や継続を値として使うためには、継続の二重化・消去が必要となることはよく知られたことである。(Scott-Strachey の表示的意味論では、現在の継続を消去し、ラベルのところで二重化して保存しておいた継続を復帰することにより、大域脱出の意味づけを行なっている。)

入力値の二重化・消去の許されない線形λ計算の関数型 $A \multimap B$ と入力値の二重化・消去が許される System F の関数型 $!A \multimap B$ との差異から考えると、継続を値と同様に使うことをできるようにするためには関数型を、

$$\begin{array}{rcl} !(B^{\perp}) \multimap ?A^{\perp} & = & ?B \wp ?A^{\perp} \\ & = & ?A^{\perp} \wp ?B \\ & = & !A \multimap ?B \end{array}$$

とすればよいはずである。

しかし、$(!A)^{\perp}$ と $?A$ とは、CUT により結合できないので、残念ながら、このような関数型では関数適用ができなくなってしまう。だが、$!A \multimap B$ ではなく、$!A \multimap !B$ [5] に対して上とおなじ変形をすると、$!A \multimap ?!B$ になる。このようにすると関数適用ができるようになる。

8.2 型から命題への変換 τ

上の議論より、$\lambda \rightarrow_c$ の型から線形論理の命題への変換は次のように定義される。

定義 5 [変換 τ] $\lambda \rightarrow_c$ の型から線形論理の命題への変換 τ を次のように (型の構造に関して) 帰納的に定義する:

$$\begin{array}{rcl} \tau(\alpha) & = & \alpha \\ \tau(A \Rightarrow B) & = & ?\tau(A)^{\perp} \wp ?!\tau(B) \\ & (= & !\tau(A) \multimap ?!\tau(B)) \end{array}$$

■

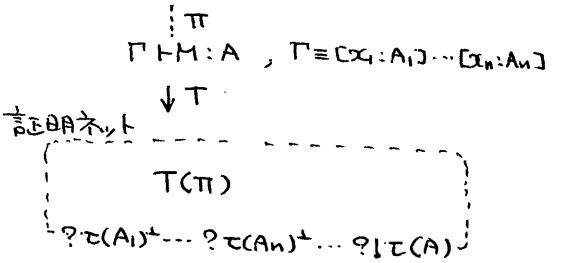
[5] 線形論理では $!A \multimap B$ の任意の証明ネットから、$!A \multimap !B$ の証明ネットを作ることができる。

8.3 項から証明への変換 T

$\lambda \rightarrow_c$ の型付けされたプログラム、すなわち、項から線形論理の証明 – 証明ネット – への変換 T を本節であたえる。

項が型付けされている時には、型付け規則をノード・葉とする木が存在するのだが、変換 T を、項からの変換として定義するよりも、このような型付け規則からなる木からの変換として定義した方が容易に定義できる。従って、本論文では型付け規則をノード・葉とする木から証明ネットへの変換として T を定義する。

変換 T は、次のような変換である。



各型付け規則に場合分けして、帰納的に定義する。(本論文の最後にまとめて載せる。)

9 継続つきプログラムとしての証明ネット

この節では、変換 T によりあたえられる証明ネットが正規化によりどのような動作をするかを述べる。

証明ネットを正規化においては次のことがわかる (正規化の例は最後にのせる)。

- β 変換において代入が実際に行われるのは、引数が変数もしくはλ抽象のときである。(すなわち、引数が値 [Plo75] のとき。)
- 脱出関数が複数あらわれたとき、最内最左のものがもっとも優先される。

これらの性質は 値呼び評価 と同様である。

10 結論

本論文では、継続つきプログラムから線形論理の証明への変換 T と プログラムの型から線形論理の命題への変換 τ とをあたえた。変換 τ によりあたえられる命題は、継続のコピー・消去が許されるという継続つきのプログラム特有の性質が自然に記述されている。そして、変換

T により与えられる証明は値呼び評価の情報も含んだようなものになっていることを指摘した。

11 関連した仕事と結論とこれからの課題

線形論理における証明の正規化にうまく対応するような計算規則をもつ継続つきのプログラミング言語を与えることは今後の最大の課題である。

Griffin [Gri90] は、古典主義的論理から直観主義的論理への埋め込みである二重否定変換と継続プリミティブをつかったプログラムから継続プリミティブを使っていないプログラムへの変換である continuation passing style transformation(CPS 変換) との同等性を議論している。

Murthy [MC90] は、二重否定変換 (= CPS 変換) 評価戦略に関する情報を付加することを指摘し、 call-by-value や call-by-name などに対応する二重否定変換を与えている。

本論文のような、線形論理による型づけは、継続のコピー・消去が許されるという継続つきのプログラム特有の性質を直接に記述しているという点で優れている。

本論文では、証明ネット自体をプログラムとして見てきたが、線形論理に基づくプログラミング言語を考えることもこれからの課題である。

参考文献

[FFKD86] Matthias Felleisen, Daniel P. Friedman, Eugene Kohlbecker, and Bruce Duba. Reasoning with continuations. In *Proceedings of the Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1986.

[FFKD87] M. Felleisen, D.P. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, Vol. 52, pp. 205–237, 1987.

[Fil89] Andrzej Filinski. Declarative continuations and categorial duality. Master's thesis, University of Copenhagen, Aug 1989. DIKU Raport Nr. 89/11, ISSN 0107-8283.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, Vol. 50, pp. 1–102, 1987.

[Gri90] Timothy G. Griffin. A formulae-as-types notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, 1990.

[GTL89] Jean-Yves Girard, .P Taylor, and Y. Lafont. *Proofs and Types*, volume 7 of *Cambridge Tracs in Computer Science*. Cambridge University Press, 1989.

[Hay86] Christopher T. Haynes. Logic continuations. In *Proceedings of the Third International Conference on Logic Programming*, pp. 671–685. Springer-Verlag, July 1986.

[HFW84] Christopher T. Haynes, Daniel P. Friedman, and Mitchell Wand. Continuations and coroutines. In *Conference Record of the 1984 ACM Symposium on Lisp and Functional Programming*, pp. 293–298, 1984.

[Laf88] Y. Lafont. The linear abstract machine. *Theoretical Computer Science*, Vol. 59, pp. 157–180, 1988.

[MC90] Chetan Murth and Robert L. Constable. Finding computational content in classical proofs, July 1990.

[Plo75] G.D. Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, Vol. 1, pp. 125–159, 1975.

[RC86] J. Rees and W. Clinger. Revised^3 repord on the algorithmic language scheme. *SIGPLAN Notices*, Vol. 21, No. 12, pp. 37–79, 1986.

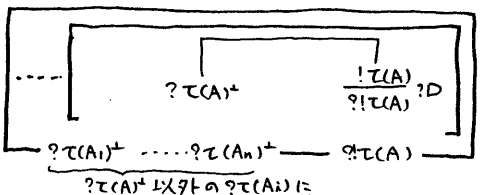
[Sto77] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[Wan78] Mitchell Wand. Continuation-based program transformation strategies. *Journal of the ACM*, Vol. 27, No. 1, pp. 174–180, 1978.

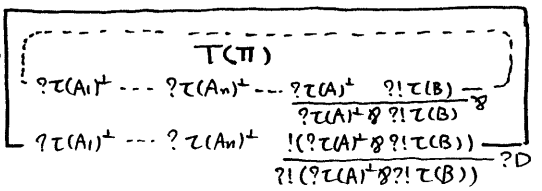
変換 T の定義

$\Gamma \equiv [x_1 : A_1] \cdots [x_n : A_n]$ とする。

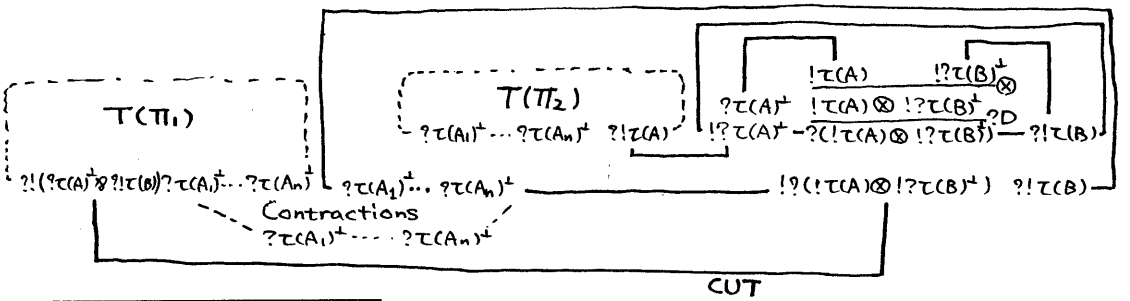
$$\frac{[x : A] \in \Gamma}{\Gamma \vdash x : A}$$ 変数規則



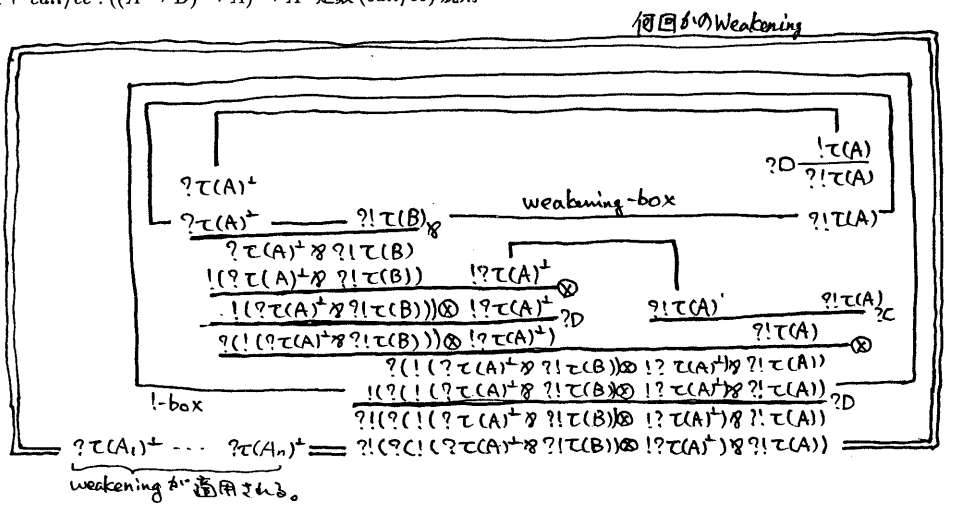
$$\frac{\begin{array}{c}\vdots \Pi \\ \Gamma[x : A] \vdash M : B\end{array}}{\Gamma \vdash (\lambda x : A \,.\, M) : A \to B}$$ λ 抽象規則



$$\frac{\begin{array}{c}\vdots \Pi_1 \\ \Gamma \vdash M : (A \to B)\end{array} \quad \begin{array}{c}\vdots \Pi_2 \\ \Gamma \vdash N : A\end{array}}{\Gamma \vdash (M\ N) : B}$$ 関数適用規則

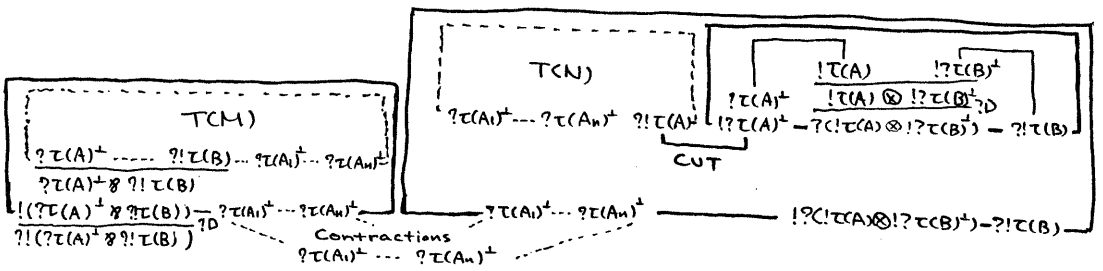


$\Gamma \vdash call/cc : ((A \to B) \to A) \to A$ 定数 $(call/cc)$ 規則

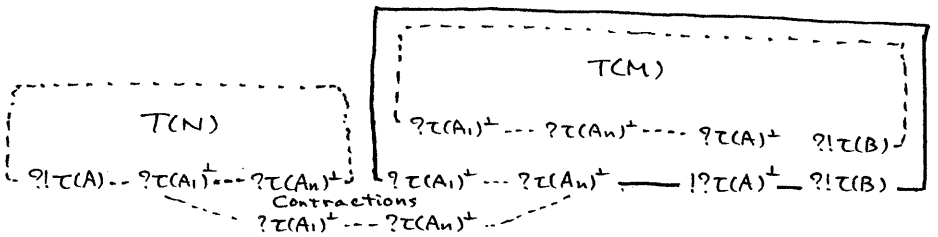


β-基

β-基 $(\lambda x : A\,.\,M^B)N^A$ から変換 T により次の証明ネットが得られる。



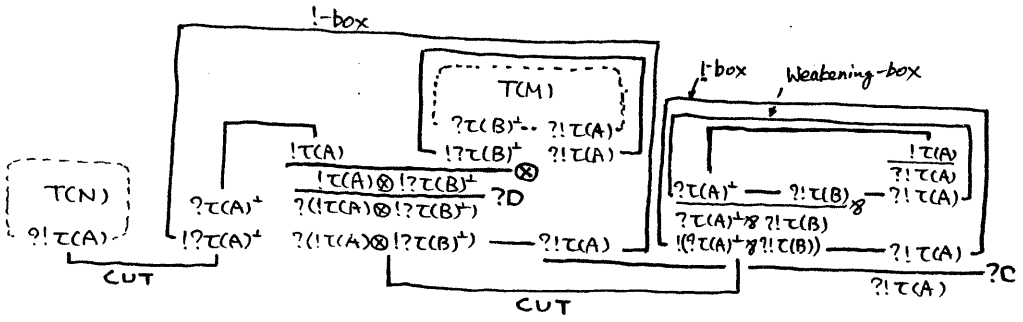
これに正規化をほどこすと次のとおり。



これが、さらに正規化がすすむためには、証明ネットの $T(N)$ の $?!\tau(A)$ に至る直前に !-box、$?C$- 規則が適用されてなければならない。そのためには、項 N が変数、λ 抽象、定数 $call/cc$ のいずれかでなくてはならないことがわかる。

$call/cc$

代表的な例として、項 $(call/cc(\lambda\, exit.(A \to B)\,.\,((\lambda x : B.M^A)(exit\ N^A))))$ について考える。
(簡単のため、自由変数に対する命題 $(?\tau(A_1^\perp)\cdots?\tau(A_n^\perp)$ など) は省略する。)
変換 T を施したのちに、β-基 $((\lambda x : B.M^A)(exit\ N^A))$ に対応した部分を正規化すると次のとおり。



これにさらに正規化の手続きをおこなうと次のとおり。