メタプログラミングのための
時相論理に基づく型付 λ 計算

湯瀬 芳洋　　五十嵐 淳
京都大学 大学院情報学研究科

{yuse,igarashi}@kuis.kyoto-u.ac.jp

概 要

メタプログラミングとは，(多段階) 部分計算や動的コード生成など，プログラムコードを計算対象のデータとして扱うパラダイムの総称であり，プログラムの効率や保守性の面で有用である．本研究では，メタプログラミングのための静的型システム設計の基礎となる型付 λ 計算 $\lambda^{\bigcirc\Box}$ を提案する．$\lambda^{\bigcirc\Box}$ は Curry-Howard 同型対応の拡張によって必然様相を含む線形時間時相論理に対応し，特性の異なる二種類のコード型を表現できる．本稿では，$\lambda^{\bigcirc\Box}$ の形式的定義を与え，subject reduction・合流性・強正規化性などの基本的な性質を証明する．また，プログラムが束縛時の順序に従って実行できるという時刻順正規化性については，先行研究ではやや非形式的に扱われていたが，より簡潔かつ一般的な形式化を行い，その厳密な証明を与える．

1　はじめに

背景　メタプログラミングとは，複数の入力を取るプログラムと入力の一部のデータから，その入力に特化したプログラムを生成する部分計算や，プログラムの実行中に新たなコードを生成・実行する実行時コード生成といった，プログラムコードを計算対象のデータとして扱うパラダイムの総称である．メタプログラミング技術は，汎用的なプログラムから状況に特化したプログラムを構成したり，システムを止めることなく新たなコードを実行したりすることを可能にし，ソフトウェアの保守性・効率性・可用性の向上に寄与すると期待できる．

メタプログラミングを効果的に支援するためには，プログラミング言語のレベルで，コードを構成・加工するための機構や，構成されたコードを実行するための機構を備えることが望ましい．このような機構を備えた言語の例としては，Lisp が挙げられる．Lisp では「‘」(擬似引用) や「,」(引用への埋めこみ) などの，マクロ定義などでコードの構成を容易にする機構や `eval` 関数という S 式で表現されたコードを実行する関数が用意されている．擬似引用は引用の拡張で，その内部で式に , を付加して記述することで，その式の評価結果を引用に埋めこむことができる．例えば `x` が `2` に束縛されている時に `‘(1 ,(+ x 1) 4)` の評価結果はリスト `(1 3 4)` になる．また Perl や Ruby にも，文字列をスクリプトと見なして実行する `eval` が見られる．しかし，生成されるコードが無事に実行できるかどうかは，大抵，実行時検査に任されており，プログラムの安全性の面からは不十分である．

本研究の目標　我々は，プログラムが生成するコードが安全に実行できるものかどうかを，なるべく早い段階—コード生成時よりも早い，コード生成を行うそもそものプログラムの実行前—に検査することが重要であると考え，安全なメタプログラミング言語のための型理論を構築することを目標としている．メタプログラミングに関わる安全性としては，文字列に対して乗算を行わない，などの一般の型システムで保証する型安全性だけではなく，有効範囲にない変数やプログラム特化後には利用できない変数に依存したコードを生成・実行しない，といった性質も保証することが望ましい．

Curry-Howard 同型対応と様相論理　このような型理論構築のための指針として，我々は，形式的論理の証明システムと型付 λ 計算との間で諸概念が密接に対応しているという Curry-Howard の同型対応の原理を利用する．Curry-Howard 同型対応は formulae-as-types 対応としても知られているもので，論理体系の論理式が型付 λ 計算の型に，論理式の証明が対応する型を持つ項に対応するというものである．様々な論理体系に対して，対応する型付 λ 計算が考えられることが知られており，近年，様相論理や時相論理と実行時コード生成・部分計算のための計算体系の対応が指摘されている [3, 4]．Davies と Pfenning による $\lambda^{\Box}$ [4] は，S4 様相論理 (の必然様相に関する部分体系) に基づく計算体系である．$\Box A$ という「必然的に A である」という論理式と「(実行時コード生成によりいつでも実行できる) A 型のコード」の型を対応させており，Lisp の「'」と `eval` に類似する機構が備わっている．この体系では，安全に `eval` を実行するために，扱うコードを自由変数がない (すなわち，実行した時に unbound variable エラーが決して起きない) ものに限定している．

一方，Davies による $\lambda^{\bigcirc}$ [3] は，離散的な時刻を考え,「次の時刻で A が成り立つ」という意味の論理式 $\bigcirc A$ などを扱うことができる線形時間時相論理に基づく計算体系である．これは，オフライン部分計算における，束縛時解析という，プログラム中のどの部分が部分計算時に計算でき (これを「(この部分の) 束縛時が静的である」という)，どの部分ができない (「束縛時が動的である」といい，この部分は剰余プログラムとして残される) かを判断するプログラム解析と密接に関連している．$\lambda^{\bigcirc}$ は，この束縛時の概念を静的/動的の二段階から多段階に一般化したような束縛時解析の型システム [6] に対応しており，$\bigcirc A$ という論理式は「次の束縛時の A 型の部分プログラム」の型に対応する．部分計算時には，束縛時が動的である部分は評価の対象ではなくコード片として扱われるので，この型は「次の束縛時で実行される A 型のコード」とも考えることができる．実際，$\lambda^{\bigcirc}$ の形式化は，静的な文脈で動的なコードを (擬似) 引用するためと，静的に計算される結果をコード内に埋め込むための，それぞれ Lisp の「`」「,」に類似した機構を備えている．この体系で扱うコードには $\lambda^{\Box}$ にみられるような制限はないが，その反面，`eval` など動的コード生成に関する機構は考えられていない．

本研究の概要　本研究で構築する型システムは,「次の時刻で A である」という様相だけではなく「今以降いつでも A である」といった様相のある線形時間時相論理の証明システムに対応しており，計算体系として $\lambda^{\Box}, \lambda^{\bigcirc}$ [3, 4] を部分体系として含むものである．これらの様相を組み合わせることにより，ある束縛時以降いつでも実行できるコードなどを体系上で表現し，その使用が安全なものかを型システムで検査することが可能になる．本研究の貢献は以下の通りである．

- 上に述べたような線形時間時相論理に対応する型付 λ 計算 $\lambda^{\bigcirc\Box}$ の文法，型システム，簡約関係を含む形式的定義を与えた．
- $\lambda^{\bigcirc\Box}$ が subject reduction，合流性，強正規化性などの基本的な性質を満たすことの証明を与えた．
- 項の簡約が束縛時の「時刻順」に行えるという，時刻順正規化性 (time-ordered normalization) [3] に関しては，可換変換 (commuting conversion) に類する簡約規則を導入することでこの性質が満たされることを示した．また，時刻順正規化性そのものについても，従来の議論に比べて，より簡潔で一般的な形式化を行い，その厳密な証明を示した．

本稿の構成　まず 2 節にて，提案する体系 $\lambda^{\bigcirc\Box}$ の概要を述べ，3 節にて，その形式的定義を与える．4 節で $\lambda^{\bigcirc\Box}$ の諸性質の証明を示した後，5 節で関連研究との比較を行い，6 節で結論を述べる．

2　$\lambda^{\bigcirc\Box}$ の概要

この節では，次節で導入する計算体系 $\lambda^{\bigcirc\Box}$ の概要，特にメタプログラミングのための機構や型システムについて，論理体系との対応の観点を交えて述べていく．

必然様相をもつ線形時間時相論理から $\lambda^{\bigcirc\Box}$ へ　前節で触れたように，$\lambda^{\bigcirc\Box}$ は Curry-Howard 同型対応により，必然様相をもつ線形時間時相論理に対応するように構築する．つまり「次の時刻で A である」という意味の $\bigcirc A$ に加えて「今以降いつでも A である」という意味の $\Box A$ という，ふたつの様相演算子を備えた論理を元にする．このような様相をとりいれた論理は，大きくわけて二種類の判断「時刻 n で A が成立する」と「時刻 n 以降でいつでも A が成立する」という判断を扱うため，[10] に倣い，証明規則で扱う仮定付判断の形式は,「時刻 n_i 以降で A_i が成り立ち，時刻 m_j で (のみ) B_j が成り立つという仮定のもとで C が時刻 n で成立する」という意味の

$$A_1^{n_1}, \ldots, A_k^{n_k}; B_1^{m_1}, \ldots, B_l^{m_l} \vdash^n C$$

というものを考える．例えば，二種類の仮定に対し，その意味から

$$\frac{n_i \leq m}{\ldots, A_i^{n_i}, \ldots; \Gamma \vdash^m A_i} \qquad \frac{}{\Delta; \ldots, B_i^m, \ldots \vdash^m B_i}$$

という仮定参照の規則が考えられる．左側の規則では，A_i が時刻 n_i 以降いつでも有効な仮定であることが，判断の結論部の時刻 m に関する条件 $n_i \leq m$ に反映され，右側の規則では，B_i が時刻 m で (のみ) の仮定であることが，仮定と $\vdash$ の肩の時刻が一致していることに反映されている．

このような論理との計算体系の対応付けを考えるための主要なアイデアは,

- 線形に順序付けられた時刻は，計算体系では束縛時に対応すると考える．
- 論理式 $\bigcirc A$ は「次の束縛時で計算できる (A 型の) **コード**の型」に対応すると考える．
- 論理式 $\Box A$ は「現束縛時とそれ以降のどの束縛時でも計算できる (A 型の) **永続的コード**の型」に対応すると考える．

という三点である．このアイデアと上の証明システムにおける判断の形式から，$\lambda^{\bigcirc\Box}$ では，変数 (仮定に対応する) として通常変数 x と，永続コードのみに束縛される永続変数 u という二種類の変数を導入することになる．そして，型判断として「永続変数 u_i が束縛時 n_i 以降で A_i 型を持ち，通常変数 x_j が束縛時 m_j で B_j 型を持つとき，項 M は束縛時 n で C 型を持つ」という意味の

$$u_1 ::^{n_1} A_1, \ldots, u_k ::^{n_k} A_k; x_1 :^{m_1} B_1, \ldots, x_l :^{m_l} B_l \vdash^n M : C$$

という形式を考えていく．

コードのための型付け規則　含意の導入・除去規則が関数抽象と関数適用 (の型付け規則) に対応するのと同様に，様相演算子の導入・除去規則がコードを構成・使用するコンストラクト (の型付け規則) に対応する．

$\bigcirc A$ は「次の時刻で A が成立する」という意味であるので「時刻 $n+1$ で A が成立するなら時刻 n で $\bigcirc A$ が成立する」「時刻 n で $\bigcirc A$ が成立するなら時刻 $n+1$ で A が成立する」という導入/除去規則が考えられる．これらに対応して,

$$\frac{\Delta; \Gamma \vdash^{n+1} M : A}{\Delta; \Gamma \vdash^n \mathbf{next}\, M : \bigcirc A} \qquad \frac{\Delta; \Gamma \vdash^n M : \bigcirc A}{\Delta; \Gamma \vdash^{n+1} \mathbf{prev}\, M : A}$$

という型付け規則が考えられる．また，導入・除去規則が続けて使われている証明が簡約基に対応するので，$\mathbf{prev}(\mathbf{next}\, M) \longrightarrow M$ という簡約が考えられる．**next**, **prev** をプログラム・コンストラクトとして考えると，それぞれ Lisp の擬似引用「‘」と擬似引用への埋め込み「,」に対応すると考えられる．例えば,

$$\underline{(\lambda z.\, \mathbf{next}(x + \mathbf{prev}\, z))\ (\mathbf{next}\, 1)} \longrightarrow_\beta \mathbf{next}(x + \underline{\mathbf{prev}\, \mathbf{next}\, 1}) \longrightarrow_\bigcirc \mathbf{next}(x+1)$$

のような簡約列が考えられる (下線は簡約基である) が，最初の項は，Lisp で記述すると，`((lambda (z) '(+ x 'z)) '1)` であり，簡約の経過は，引用された S 式が「,」の部分で引用がキャンセルされ外側の擬似引用に埋め込まれることに対応している．

一方,「今以降いつでも A 」という論理式 $\Box A$ は，ある特定の時刻に依存した仮定をすることなく A が証明できれば，導入することができる．これは，永続的なコードは複数の束縛時で実行される可能性があるため，ある束縛時でしか有効ではない通常の変数を (自由変数として) 含んでいてはいけない，ということに対応する．このことから，

$$\frac{\Delta;\cdot \vdash^n M : A}{\Delta;\Gamma \vdash^n \mathbf{box}\, M : \Box A} \qquad \frac{\Delta;\Gamma \vdash^n M : \Box A \qquad \Delta, u::^n A;\Gamma \vdash^n N : B}{\Delta;\Gamma \vdash^n \mathbf{let\ box}\, u = M\ \mathbf{in}\ N : B}$$

という型付け規則が考えられる．左側の (導入) 規則の前提では，空の通常変数コンテキスト「$\cdot$」によって自由変数がないことを示している (ただし，n で有効な永続変数は，任意の $m \geq n$ なる m でも有効であるため，自由変数として含まれていてもよい)．また，右側の (除去) 規則では，導かれた $\Box A$ を，永続的な仮定として使用する規則になっている．**box** は永続的なコードを構成するコンストラクトであり，$\mathbf{let\ box}\, u = M\ \mathbf{in}\ N$ は M を計算して得た永続的コード $\mathbf{box}\, M'$ を分解して u を M' に束縛して，N を計算するコンストラクトと考えられる．すなわち，$\mathbf{let\ box}\, u = \mathbf{box}\, M\ \mathbf{in}\ N \longrightarrow [M/u]N$ という簡約規則をもつ ($[M/u]$ は永続変数への代入操作である)．$\Box A$ は「今以降いつでも A」という意味なので，ただの A も含意しているが，これは永続コードを即実行 (`eval`) できることを示している．例えば，以下の簡約列は，1 を足す関数のコードを実行して 2 に適用するプログラムの実行を表現している．

$$\underline{\mathbf{let\ box}\, u = \mathbf{box}(\lambda x.\, x+1)\ \mathbf{in}\ u\ 2} \longrightarrow_{\Box} [(\lambda x.\, x+1)/u](u\ 2) \equiv \underline{(\lambda x.\, x+1)\ 2} \longrightarrow_{\beta} 2+1$$

このように，ふたつの様相演算子をコードの二種類の型と解釈することで，時相論理に対応する型付計算体系を構築していく．

let box の一般化　我々は $\lambda^{\bigcirc\Box}$ を構築するにあたって，メタプログラミングのための表現力と同様に，自然演繹体系としての表現力も重視して設計を行なった．つまり，セマンティクスや公理系の観点から等価な論理式同士は，それが等価であることを両方向の含意として証明できるべきだと考えた．例えば，$\bigcirc\Box A$ と $\Box\bigcirc A$ はともに，現在時刻が 0 であるとすると，時刻 1，時刻 2，... で A が成立することを意味する論理式であるため，$\bigcirc\Box A \to \Box\bigcirc A$ かつ $\Box\bigcirc A \to \bigcirc\Box A$ が証明可能であるべきである．これが成立すれば，($\Box A$ と $\Box\Box A$ が等価であることも併せると)，全ての論理式は $\underbrace{\bigcirc\cdots\bigcirc}_{n}\Box A$ もしくは $\underbrace{\bigcirc\cdots\bigcirc}_{n} A$ (ただし $n \geq 0$) と論理的に等価であることが示せ，計算体系としても，永続的コードは「数ステージ先になると永続的に使用できるようになるコード」という標準的な形にすることができる．

問題となる命題のうち，後者の $\Box\bigcirc A \to \bigcirc\Box A$ については，これまでに説明した型付け規則を用いて，

$$\lambda x{:}\Box\bigcirc A.\, \mathbf{let\ box}\, u = x\ \mathbf{in}\ \mathbf{next\, box\, prev}\, u$$

という項が $\Box\bigcirc A \to \bigcirc\Box A$ という型を持つ，つまり「証明」になっていることが示せる．しかし，前者の $\bigcirc\Box A \to \Box\bigcirc A$ は，これまでに導入した規則では証明できそうもないことがわかった．

そこで，この問題を解消するため，$\lambda^{\bigcirc\Box}$ では **let box** 式の構文 $\mathbf{let\ box}\, u = M\ \mathbf{in}\ N$ において，仮定の束縛 $\mathbf{box}\, u = M$ に，本体式 N と異なる束縛時を許容することとした．具体的には，$\mathbf{let\ box}\, u =_i M\ \mathbf{in}\ N$ という構文に改め，$\mathbf{box}\, u =_i M$ が N より i 時刻先の束縛時に属することを示すものとし，

$$\frac{\Delta;\Gamma \vdash^{n+i} M : \Box A \qquad \Delta, u::^{n+i} A;\Gamma \vdash^n N : B}{\Delta;\Gamma \vdash^n \mathbf{let\ box}\, u =_i M\ \mathbf{in}\ N : B}$$

という型付け規則を考える．この規則により，命題 $\bigcirc\Box A \to \Box\bigcirc A$ は，

$$\lambda x{:}\bigcirc\Box A.\,\mathsf{let\ box}\,u =_1 \mathsf{prev}\,x\ \mathsf{in\ box\,next}\,u$$

という項で「証明」することができる．

時刻順正規化性と交換規則の導入　メタプログラミングにおいては，一般にプログラム中に様々な束縛時の (部分) 項が混在しており，実行モデルを考えるには，それらの実行順序を決定することが必要である．そのための指針となるのが，β 簡約が束縛時の「時刻順」に行うことができるという，時刻順正規化性 (time-ordered normalization) [3] と呼ばれる性質である．時刻順正規化性は，計算を進めるにあたって，過去の束縛時に遡る必要がなく正規化できることを保証しており，計算体系が束縛時を正しく表現することの裏付けとなるものでもある．

たとえば，次の簡約例では，β 簡約が束縛時の小さい順に行われており，時刻順正規化性が成り立っている．

$$\underline{(\lambda x.\,\mathsf{next}((\mathsf{prev}\,x)\,y))\,\mathsf{next}\,\lambda z.\,z} \xrightarrow{0}_\beta \mathsf{next}((\underline{\mathsf{prev\,next}\,\lambda z.\,z})\,y) \longrightarrow_\bigcirc \mathsf{next}(\underline{(\lambda z.\,z)\,y}) \xrightarrow{1}_\beta \mathsf{next}\,y$$

ここで，$\longrightarrow$ の上に付加した数字は，β 簡約の時刻である．例えば **next** 内の簡約基は，その外側よりも時刻が遅いと考えられる．

しかし，$\lambda^{\bigcirc\Box}$ では，前項で述べたように **let box** 式において異なる束縛時の仮定を許容する規則を採用したため，標準的な簡約規則だけでは，時刻順正規化性は一般に成立しない．たとえば，次の反例では，可能な唯一の簡約列において，時刻 1 で簡約を行った後に，過去の束縛時 (0 時) の β 基が出現してしまっている．

$$(\mathsf{let\ box}\,u =_1 \underline{(\lambda x.\,x)\,\mathsf{box}\,t}\ \mathsf{in}\ \lambda y.\,y)\,z \xrightarrow{1}_\beta \underline{(\mathsf{let\ box}\,u =_1 \mathsf{box}\,t\ \mathsf{in}\ \lambda y.\,y)}\,z \longrightarrow_\Box \underline{(\lambda y.\,y)\,z} \xrightarrow{0}_\beta z$$

この問題を解消するために，$\lambda^{\bigcirc\Box}$ では，標準的な簡約規則に加えて，以下の可換変換を含む簡約システムを考える．

$$(\mathsf{let\ box}\,u =_i P\ \mathsf{in}\ Q)\,R \longrightarrow_{\mathrm{com}} \mathsf{let\ box}\,u =_i P\ \mathsf{in}\ (Q\,R)$$
$$\mathsf{prev}(\mathsf{let\ box}\,u =_{i+1} P\ \mathsf{in}\ Q) \longrightarrow_{\mathrm{com}} \mathsf{let\ box}\,u =_i P\ \mathsf{in}\ (\mathsf{prev}\,Q)$$
$$\mathsf{let\ box}\,t =_i (\mathsf{let\ box}\,u =_j P\ \mathsf{in}\ Q)\ \mathsf{in}\ R \longrightarrow_{\mathrm{com}} \mathsf{let\ box}\,u =_{i+j} P\ \mathsf{in}\ (\mathsf{let\ box}\,t =_i Q\ \mathsf{in}\ R)$$

これらの規則は，**let box** 式の本体の範囲を拡大して，簡約の可能性を拡げるものであり，直和型を導入した型付 λ 計算体系に見られる可換変換 (commuting conversion) [5] に類似するものである．

このような簡約システムを仮定すると，前述の反例は，次のように「時刻順」に簡約することが可能になる．

$$\underline{(\mathsf{let\ box}\,u =_1 (\lambda x.\,x)\,\mathsf{box}\,t\ \mathsf{in}\ \lambda y.\,y)\,z} \longrightarrow_{\mathrm{com}} \mathsf{let\ box}\,u =_1 (\lambda x.\,x)\,\mathsf{box}\,t\ \mathsf{in}\ \underline{(\lambda y.\,y)\,z}$$
$$\xrightarrow{0}_\beta \mathsf{let\ box}\,u =_1 \underline{(\lambda x.\,x)\,\mathsf{box}\,t}\ \mathsf{in}\ z \xrightarrow{1}_\beta \underline{\mathsf{let\ box}\,u =_1 \mathsf{box}\,t\ \mathsf{in}\ z} \longrightarrow_\Box z$$

以上を元に次節以降では，$\lambda^{\bigcirc\Box}$ の形式的定義を与え，その型付計算体系としての諸性質を考察していく．

3　体系 $\lambda^{\bigcirc\Box}$ の形式的定義

3.1　構文規則

時刻 (または**束縛時**) を 0 以上の自然数とし，メタ変数 m, n などによって表す．また， $\mathbf{OVars}$ と $\mathbf{PVars}$ を，それぞれ可算無限な**通常変数**と**永続変数**の集合とし，それぞれ，メタ変数 x, y などとメタ変数 t, u などによってその要素を表す．ここで，$\mathbf{PVars} \cap \mathbf{OVars} = \emptyset$ であるとする．さらに，メタ変数 b によって**基底型**を表し，その全体を $\mathbf{BTypes}$ とする．

$\lambda^{\bigcirc\Box}$ の **型**，**項** (または**式**) を，次のように定める．

$$\text{型}\quad A, B \;::=\; b \mid A \to B \mid \bigcirc A \mid \Box A$$

$$\text{項}\quad M, N \;::=\; x \mid u \mid \lambda x{:}A.\, M \mid M\ N \mid \mathsf{next}\, M \mid \mathsf{prev}\, N \mid \mathsf{box}\, M \mid \mathsf{let\ box}\, u =_i M\ \mathsf{in}\ N$$

let box 式における $=$ の添字の i は任意の自然数である．$i = 0$ の場合は，添字を省略して $\mathsf{let\ box}\, u = M\ \mathsf{in}\ N$ と略記する．$\bigcirc$, $\Box$ は $\to$ よりも強く結合し，**let box** 式は λ と同様，できるだけ右に伸びるように読むことにする．つまり，$\bigcirc A \to B$ は $(\bigcirc A) \to B$ のことであり，$\mathsf{let\ box}\, u =_i M\ \mathsf{in}\ x\ y$ は $\mathsf{let\ box}\, u =_i M\ \mathsf{in}\ (x\ y)$ のことである．

$\lambda x{:}A.\, M$ の x と $\mathsf{let\ box}\, u =_i N\ \mathsf{in}\ M$ の u は，M を有効範囲とする束縛変数である．これに従い M の自由永続変数 $\mathrm{FPV}(M)$，自由通常変数 $\mathrm{FOV}(M)$，自由変数 $\mathrm{FV}(M) = \mathrm{FPV}(M) \cup \mathrm{FOV}(M)$ を，通常の λ 計算と同様に定義する．たとえば，$M \equiv \mathsf{let\ box}\, u = \mathsf{box}(\lambda x{:}A.\, x)\ \mathsf{in}\ u\ z$ とすると，$\mathrm{FPV}(M) = \emptyset$, $\mathrm{FOV}(M) = \{z\}$ となる．以降では，α 同値による暗黙の束縛変数の名前替えを仮定し，束縛変数に関しては，互いにすべて異なり，また自由変数とも異なるとする．

また，自由変数の捕捉を回避した，通常/永続変数への項の代入 $[M/x]N$ および $[M/u]N$ また，項の α 同値関係 $M \equiv_\alpha N$ についても，通常の λ 計算と同様に定義する．

3.2 型システム

前節で述べたように，$\lambda^{\bigcirc\Box}$ の型判断は，$\Delta; \Gamma \vdash^n M : A$ によって表し，項 M は，時刻 n において，永続コンテキスト Δ および通常コンテキスト Γ のもとで，型 A をもつことを意味するものとする．ここで，永続コンテキストおよび通常コンテキストは，次のように定義される．

$$\text{永続コンテキスト}\quad \Delta \;::=\; \cdot \mid \Delta, u{::}^n A$$
$$\text{通常コンテキスト}\quad \Gamma \;::=\; \cdot \mid \Gamma, x{:}^n A$$

これらのコンテキストは，いずれも永続変数または通常変数に関する仮定を列挙したものであり，それぞれの仮定の直観的な意味は，次のとおりである．すなわち，$u{::}^n A$ は，永続変数 u が時刻 n 以降で有効な A 型の変数であることを宣言し，$x{:}^n A$ は，通常変数 x が時刻 n でのみ有効な A 型の変数であることを宣言する．以下では，ひとつのコンテキストに宣言されている変数には重複がないと仮定する．

上記の型判断を導出するための型付け規則を，次のように定める．

$$(\textsc{t-OVar})\ \frac{(x{:}^n A \in \Gamma)}{\Delta; \Gamma \vdash^n x : A} \qquad (\textsc{t-PVar})\ \frac{(u{::}^m A \in \Delta \qquad m \le n)}{\Delta; \Gamma \vdash^n u : A}$$

$$(\textsc{t-Abs})\ \frac{\Delta; \Gamma, x{:}^n A \vdash^n M : B}{\Delta; \Gamma \vdash^n \lambda x{:}A.\, M : A \to B} \qquad (\textsc{t-App})\ \frac{\Delta; \Gamma \vdash^n M : A \to B \qquad \Delta; \Gamma \vdash^n N : A}{\Delta; \Gamma \vdash^n M\ N : B}$$

$$(\textsc{t-Next})\ \frac{\Delta; \Gamma \vdash^{n+1} M : A}{\Delta; \Gamma \vdash^n \mathsf{next}\, M : \bigcirc A} \qquad (\textsc{t-Prev})\ \frac{\Delta; \Gamma \vdash^n M : \bigcirc A}{\Delta; \Gamma \vdash^{n+1} \mathsf{prev}\, M : A} \qquad (\textsc{t-Box})\ \frac{\Delta; \cdot \vdash^n M : A}{\Delta; \Gamma \vdash^n \mathsf{box}\, M : \Box A}$$

$$(\textsc{t-LetBox})\ \frac{\Delta; \Gamma \vdash^{n+i} M : \Box A \qquad \Delta, u{::}^{n+i} A; \Gamma \vdash^n N : B \qquad (0 \le i)}{\Delta; \Gamma \vdash^n \mathsf{let\ box}\, u =_i M\ \mathsf{in}\ N : B}$$

規則 (t-OVar) と (t-PVar) は前節の最初で述べた仮定参照の証明規則に対応するものである．(t-OVar), (t-Abs), (t-App) の各規則は，変数や判断の束縛時 n が全て一致していなければならないという制約を除いて，単純型付 λ 計算における変数参照・関数抽象・関数適用の各規則と同様である．

規則 (t-Next) と (t-Prev) は，次の時刻で実行できるコードの生成と評価に対応する．規則 (t-Next) は，ある時刻で A 型をもつ項を，その前の時刻で $\bigcirc A$ 型をもつコード式として参照することができることを

示している．逆に，規則 (T-Prev) は，ある時刻で $\bigcirc A$ 型をもつコード式を，次の時刻で評価して A 型の項を得ることができることを示している．

規則 (T-Box) と (T-LetBox) は，現在の時刻以降常に実行できる永続的コードの生成と実行に対応する．先に述べたように，規則 (T-Box) の前提においては，通常コンテキストは空でなければならない．規則 (T-LetBox) は，項 N が依存する仮定 $u{::}^{n+i}A$ を，同じ時刻 $n+i$ で型 $\Box A$ をもつコード式 M で置きかえることができることを示している．この時刻は，N の時刻 n と異なっていてもよく，一般には N より未来の時刻に属する仮定を許容する．すなわち，**let box** 式における $=$ の添字の i は，本体式 N と束縛 $\mathbf{box}\, u = M$ の時刻の差を示す．

項 M が $\Delta;\Gamma \vdash^n M : A$ と型付けされるとき，M の**項時刻** (term time) は n である，と定義する．より一般的に，M の部分項の項時刻を，次のように定める．項 M が，型導出 $\mathcal{D}$ によって $\Delta;\Gamma \vdash^n M : A$ と型付けされるとし，$\mathcal{D}$ の部分導出によって M の部分項 M' が $\Delta';\Gamma' \vdash^{n'} M' : A'$ と導出されるとする．このとき，M' **は** $\mathcal{D}$ **において項時刻** n' **をもつ**，あるいは，($\mathcal{D}$ が文脈から明らかなときや，$\mathcal{D}$ を暗黙のうちに仮定するときは) 単に，M' は時刻 n' をもつ，という．項時刻は，その項が属する束縛時を表すものである．

たとえば，$M \equiv \mathbf{prev}(\mathbf{let\ box}\, u =_{i+1} \mathbf{box}\, P\ \mathbf{in}\ Q)$, $\Delta;\Gamma \vdash^1 M : A$ とすると，M, P, Q の項時刻は，それぞれ 1, $i+1$, 0 である．

3.3 簡約システム

$\lambda^{\bigcirc\Box}$ の**簡約関係**を，$M \xrightarrow{\ell} N$ によって表し，項 M は，時刻 ℓ において，項 N に 1 ステップで簡約されることを意味するものとする．ℓ は，簡約が行われる束縛時 (**簡約時刻** (reduction time)) を表わすが，時刻を特に問題にしないときは，単に $M \longrightarrow N$ と書く．また，関係 $\longrightarrow$ の反射的推移的閉包を $\longrightarrow^*$ によって表す．

簡約関係を導くための簡約規則を，次のように定める．いずれの規則も，簡約時刻 ℓ は，それぞれ下線で示した部分項の項時刻と一致しているものとする．任意の部分式で簡約を行える full reduction を仮定するが，いわゆる合同規則 (congruence rule) は省略している．なお，合同規則は簡約時刻を保存する．

$$
\begin{array}{lll}
\text{(R-Beta)} & \underline{(\lambda x{:}A.\, M)}\ N \xrightarrow{\ell} [N/x]M & \\
\text{(R-Prev)} & \mathbf{prev}\,\underline{(\mathbf{next}\, M)} \xrightarrow{\ell} M & \\
\text{(R-LetBox)} & \mathbf{let\ box}\, u =_i \underline{\mathbf{box}\, M}\ \mathbf{in}\ N \xrightarrow{\ell} [M/u]N & \\
\text{(R-com-App)} & \underline{(\mathbf{let\ box}\, u =_i L\ \mathbf{in}\ M)}\ N \xrightarrow{\ell} \mathbf{let\ box}\, u =_i L\ \mathbf{in}\ (M\ N) & (u \notin \mathrm{FPV}(N)) \\
\text{(R-com-Prev)} & \mathbf{prev}\,\underline{(\mathbf{let\ box}\, u =_{i+1} L\ \mathbf{in}\ M)} \xrightarrow{\ell} \mathbf{let\ box}\, u =_i L\ \mathbf{in}\ (\mathbf{prev}\, M) & \\
\text{(R-com-LetBox)} & \mathbf{let\ box}\, t =_i \underline{(\mathbf{let\ box}\, u =_j L\ \mathbf{in}\ M)}\ \mathbf{in}\ N & \\
 & \xrightarrow{\ell} \mathbf{let\ box}\, u =_{i+j} L\ \mathbf{in}\ (\mathbf{let\ box}\, t =_i M\ \mathbf{in}\ N) & (u \notin \mathrm{FPV}(N))
\end{array}
$$

(R-Beta) 規則は通常の β 簡約である．(R-Prev) 規則は **next** コード式の評価に，(R-LetBox) 規則は **box** コード式の埋め込み/実行に，それぞれ対応する．(R-com-{App,Prev,LetBox}) は，前節の最後で言及した交換規則 (commuting conversions) である．

簡約時刻は，対応する簡約が実際に行われる (べき) 束縛時を表すものである．簡約基自身の項時刻と簡約時刻は，必ずしも一致しない．

たとえば，$M \equiv \mathbf{prev}(\mathbf{let\ box}\, u =_{i+1} \mathbf{box}\, P\ \mathbf{in}\ Q)$, $\Delta;\Gamma \vdash^1 M : A$ とすると，(R-LetBox) および合同規則によって $M \xrightarrow{i+1} \mathbf{prev}([P/u]Q)$ であり，(R-com-Prev) によって $M \xrightarrow{0} \mathbf{let\ box}\, u =_i \mathbf{box}\, P\ \mathbf{in}\ \mathbf{prev}\, Q$ である．

3.4 例

$\lambda^{\bigcirc\Box}$ による記述例として，指数関数のプログラム例を挙げる．すなわち，x^n を計算するプログラムであるが，指数 n と基数 x の束縛時を分離し，n に特化したコードを生成するような状況を想定する．

以下では，これまでに定義した $\lambda^{\bigcirc\Box}$ の核言語に，局所定義のための **let** 構文，整数型 **int** やその加減乗除の演算，真偽値型 **bool** や比較演算子や **if-then-else** 構文，さらに，再帰的関数定義のための構文 $\mathbf{fix}\, f{:}A.\, M$ (M 中の束縛変数 f が $\mathbf{fix}\, f.\, M$ 自身を指す) を加えた言語を仮定している．また，可読性のため，型宣言は適宜省略する．

さらに，簡約の例については，**next** 式や **box** 式の内部は引用されたコードであると考え，**prev** によってエスケープされた時のみ計算を行うとする．つまり，別の言い方をすると，計算が部分計算のように束縛時毎に段階的に進む様子を示す．

まず，計算ステージの分離を行わない，通常の定義 `power` と，その評価例を示す．

$$
\begin{aligned}
&\texttt{power} : \mathbf{int} \to \mathbf{int} \to \mathbf{int} \\
&\quad \equiv \mathbf{fix}\, p{:}\mathbf{int}\to\mathbf{int}\to\mathbf{int}.\, \lambda n{:}\mathbf{int}.\, \mathbf{if}\ n = 0\ \mathbf{then}\ \lambda x{:}\mathbf{int}.\, 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \ \mathbf{let}\ q = p\ (n-1)\ \mathbf{in}\ \lambda x{:}\mathbf{int}.\, x * (q\ x) \\
&\texttt{power}\ 2 \longrightarrow^* \lambda x.\, x * ((\lambda x.\, x * ((\lambda x.\, 1)\ x))\ x)
\end{aligned}
$$

`power` は，引数 n のみによる部分適用が可能であり，評価例に見られるように，特定の n の値に特化した式を，関数値として返すことができる．

これを永続的なコードとして返すことができるように，**box** を用いて書き換えたものが次に定義する $\texttt{power}^{\Box}$ である．引数 n を与えると，現在の束縛時以降いつでも x の n 乗を計算できるコードを返す．

$$
\begin{aligned}
&\texttt{power}^{\Box} : \mathbf{int} \to \Box(\mathbf{int} \to \mathbf{int}) \\
&\quad \equiv \mathbf{fix}\, p{:}\mathbf{int}\to\Box(\mathbf{int}\to\mathbf{int}).\, \lambda n{:}\mathbf{int}.\, \mathbf{if}\ n = 0\ \mathbf{then}\ \mathbf{box}(\lambda x{:}\mathbf{int}.\, 1) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \ \mathbf{let}\ \mathbf{box}\, u = p\ (n-1)\ \mathbf{in}\ \mathbf{box}(\lambda x{:}\mathbf{int}.\, x * (u\ x)) \\
&\texttt{power}^{\Box}\ 2 \longrightarrow^* \mathbf{box}(\lambda x.\, x * ((\lambda x.\, x * ((\lambda x.\, 1)\ x))\ x))
\end{aligned}
$$

このようにして，いつでも実行できる自乗関数のコードが得られるが，内部に $(\lambda x.\, 1)\ x$ などの変数に適用しているだけの冗長な簡約基 (variable-for-variable redex) を含み，必ずしも効率のよいコードであるとは言えない．

$\bigcirc$ を使うと，次の定義 $\texttt{power}^{\bigcirc}$ のように，この点を改善することができる．

$$
\begin{aligned}
&\texttt{power}^{\bigcirc} : \mathbf{int} \to \bigcirc(\mathbf{int} \to \mathbf{int}) \\
&\quad \equiv \lambda n{:}\mathbf{int}.\ \mathbf{next}(\lambda x{:}\mathbf{int}.\ \mathbf{prev}(\\
&\qquad\quad (\mathbf{fix}\, p{:}\mathbf{int}\to\bigcirc\mathbf{int}.\, \lambda m{:}\mathbf{int}.\, \mathbf{if}\ m = 0\ \mathbf{then}\ \mathbf{next}\, 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathbf{else}\ \ \mathbf{next}(x * \mathbf{prev}(p\ (m-1)))) \\
&\qquad\quad n)) \\
&\texttt{power}^{\bigcirc}\ 2 \longrightarrow^* \mathbf{next}(\lambda x.\, x * (x * 1))
\end{aligned}
$$

この定義では **prev** を使ってコードから脱出することで，その外で宣言された変数 x (の引用) を使った計算をして，冗長な簡約基を解消している．$\texttt{power}^{\bigcirc}$ の定義中の $(\mathbf{fix}\, p.\, \cdots)$ は，n を受けとって，$\mathbf{next}(\underbrace{x * x * \cdots * x}_{n} * 1)$ という，x を自由に含む $\bigcirc\,\mathbf{int}$ 型のコードを返す関数である．

box の内部には自由変数を含むことができないため，**box**/**let box** 構文だけでは，$\texttt{power}^{\bigcirc}$ に相当するような，効率のよいコードを生成する関数を記述することができない．一方，$\texttt{power}^{\bigcirc}$ で生成されるコードは次の束縛時のみでしか実行できず，汎用性に欠ける．

我々が $\lambda^{\bigcirc\Box}$ で目ざした目標の一つは，**next** コード式にみられる柔軟性と，**box** コード式にみられる汎用性の高さの両方を兼ね備えたコードの表現を可能にすることであった．$\lambda^{\bigcirc\Box}$ では，次に示すように，**next** と **box** を組み合わせた $\texttt{power}^{\bigcirc\Box}$ 関数を定義することができる．

$$
\begin{aligned}
&\texttt{power}^{\bigcirc\Box} : \Box\,\mathsf{int} \to \bigcirc\,\Box(\mathsf{int} \to \mathsf{int}) \\
&\qquad \equiv \lambda n{:}'\Box\mathsf{int}.\,\mathsf{let}\ \ \mathsf{box}\,u = n'\ \mathsf{in}\ \ \mathsf{next}\,\mathsf{box}\,\lambda x{:}\mathsf{int}.\ \mathsf{prev}(\\
&\qquad\qquad (\mathsf{fix}\,p{:}\mathsf{int}{\to}\bigcirc\mathsf{int}.\,\lambda m{:}\mathsf{int}.\,\mathsf{if}\ m = 0\ \mathsf{then}\ \mathsf{next}\,1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{else}\ \ \mathsf{next}(x * \mathsf{prev}(p\ (m-1)))) \\
&\qquad\qquad u) \\
&\texttt{power}^{\bigcirc\Box}\ (\mathsf{box}\,2) \longrightarrow^* \mathsf{next}\,\mathsf{box}(\lambda x.\,x * (x * 1))
\end{aligned}
$$

評価結果のコードは，$\texttt{power}^{\bigcirc}$ と同等の効率のよいコードとなっている．また，生成コードの型が $\bigcirc\,\Box(\mathsf{int} \to \mathsf{int})$ であることから，次の束縛時以降の任意のステージにおいて実行可能なコードであり，例えば

$$
\mathsf{let}\ \ \mathsf{box}\,u =_1 \mathsf{prev}(\texttt{power}^{\bigcirc\Box}(\mathsf{box}\,3))\ \mathsf{in}\ \ \mathsf{next}(\cdots u\ 5 \cdots \mathsf{next}(\cdots u\ 9 \cdots)\cdots)
$$

のように，特化して生成した三乗関数をステージを跨って利用することができる．

ただし，この $\texttt{power}^{\bigcirc\Box}$ の定義は，以下の二点で $\texttt{power}^{\bigcirc}$, $\texttt{power}^{\Box}$ よりも不利である．

一つは，引数 n の型が int でなく $\Box\,\mathsf{int}$ となっていることである．しかし，基底型の定数 (2 や `true` や `"foo"` など) は実際には任意の束縛時で有効であるので，定数については，上の例のように，常に $\Box b$ 型を与えることが考えられる．あるいは，基底型に対して，$\mathrm{ML}^{\Box}$ [12] などに見られる **lift** オペレータとそれに関する規則，

$$
(\textsc{t-Lift})\ \frac{\Delta;\Gamma \vdash^n M : b}{\Delta;\Gamma \vdash^n \mathsf{lift}\,M : \Box\,b}
$$

を導入する方法も考えられる．**lift** は，引数の M を評価したうえで，その結果を **box** コード式として返すものである．以上のような措置を考慮すると，この点は実際上はそれほど大きな問題ではないものと思われる．

もう一つの問題点は，生成されるコードの型が $\Box(\mathsf{int} \to \mathsf{int})$ ではなく，$\bigcirc\,\Box(\mathsf{int} \to \mathsf{int})$ となっていることである．このために，このコードは次の束縛時以降では利用できるものの，現在の束縛時で即時に実行することはできない．ここで仮定している評価の方法では，**box** コードの内部で部分計算を進めるためには，**prev** によってエスケープする必要があり ($\texttt{power}^{\bigcirc\Box}$ の定義中の $\mathsf{prev}((\mathsf{fix}\,p.\,\cdots)\,u)$ の部分がこれに該当する)，項全体の時刻を考えると，どうしても **next** を用いる必要がある．実行時コード生成の観点からは，特化したコードを即使うことができず，不満の残る結果であるが，現段階では，おそらくこれが限界でないかと思われる．

4 $\lambda^{\bigcirc\Box}$ の性質

この節では，$\lambda^{\bigcirc\Box}$ の性質を述べる．一般の計算体系としての基本的な性質である，subject reduction，合流性，強正規化性，また，計算体系がメタプログラミングのための体系として適当であることを述べた，時刻順正規化性についての証明を行う．本論文では証明の概略のみを与えるに留めるが，より詳細な証明に関しては第一著者の修士論文 [13] を参照されたい．

4.1 Subject Reduction

Subject reduction は，項の型付けが，簡約の前後で変化しないことを保証する性質である．厳密には，次のように定式化される．

定理 1 (Subject Reduction) もし $\Delta;\Gamma \vdash^n M : A$ かつ $M \longrightarrow M'$ ならば $\Delta;\Gamma \vdash^n M' : A$ である．

以下では，まず，項の型付けは時刻を「進めて」も保存されるという性質 (補題 2) と，いわゆる weakening (補題 3)，また代入補題 (補題 4) を証明する．

準備として以下の定義を導入する．$\Delta \preceq \Delta'$ を，(1) $\cdot \preceq \cdot$, (2) $\Delta \preceq \Delta'$ ならば $\Delta, u::^n A \preceq \Delta', u::^n A$，(3) $\Delta \preceq \Delta'$ ならば $\Delta, u::^n A \preceq \Delta', u::^{n+1} A$，の三つの規則に関して閉じた最小の関係として定義する．また，$\Gamma \prec \Gamma'$ を，(1) $\cdot \prec \cdot$, (2) $\Gamma \prec \Gamma'$ ならば $\Gamma, x:^n A \prec \Gamma', x:^{n+1} A$，の二つの規則に関して閉じた最小の関係として定義する．

補題 2 (Time Increment) もし $\Delta; \Gamma \vdash^n M : A$ かつ $\Delta \preceq \Delta', \Gamma \prec \Gamma'$ ならば $\Delta'; \Gamma' \vdash^{n+1} M : A$ である．特に，もし $\Delta; \cdot \vdash^m M : A$ かつ $m \leq n$ ならば $\Delta; \cdot \vdash^n M : A$ である．

【証明】後者は前者より帰結するので，前者を示せば十分である．この証明は，項 M の構造に関する帰納法による．以下では，永続変数および **let box** の場合のみ示す．
《$M \equiv u$ のとき》(T-PVar) より，$m \leq n$ なる時刻 m が存在して，$u::^m A \in \Delta$ である．このとき，$u::^m A \in \Delta'$ であるか，$u::^{m+1} A \in \Delta'$ である．いずれの場合も，(T-PVar) より結論を得る．
《$M \equiv \mathbf{let\ box}\, u =_i L\ \mathbf{in}\ N$ のとき》(T-LetBox) より，ある型 B が存在して，$\Delta; \Gamma \vdash^{n+i} L : \Box B$ かつ $\Delta, u::^{n+i} B; \Gamma \vdash^n N : A$．帰納法の仮定より，$\Delta'; \Gamma' \vdash^{n+1+i} L : \Box B$ かつ $\Delta', u::^{n+1+i} B; \Gamma' \vdash^{n+1} N : A$ (T-LetBox) より結論を得る． □

次の weakening は，型付けにおいて冗長な変数仮定を許すものであるが，証明はほぼ自明なので省略する．

補題 3 (Weakening) $\mathrm{dom}(\Delta)$, $\mathrm{dom}(\Gamma)$ により，それぞれコンテキスト Δ, Γ に含まれる変数の集合を表す．

1. もし $\Delta; \Gamma \vdash^n M : A$ かつ $u \notin \mathrm{dom}(\Delta)$ ならば $\Delta, u::^m B; \Gamma \vdash^n M : A$ である．
2. もし $\Delta; \Gamma \vdash^n M : A$ かつ $x \notin \mathrm{dom}(\Gamma)$ ならば $\Delta; \Gamma, x:^m B \vdash^n M : A$ である．

次の代入補題は，変数を同じ型と束縛時をもつ項で置きかえても，全体の型付けが変化しないことを示す．ただし，永続変数に関しては，それが動く範囲から，自由変数を含まない項，すなわち，空の通常コンテキストのもとで型付け可能なものの代入に限られる．

補題 4 (代入補題 (Substitution Lemma))

1. もし $\Delta; \Gamma, x:^m B \vdash^n M : A$ かつ $\Delta; \Gamma \vdash^m N : B$ ならば $\Delta; \Gamma \vdash^n [N/x]M : A$ である．
2. もし $\Delta, u::^m B; \Gamma \vdash^n M : A$ かつ $\Delta; \cdot \vdash^m N : B$ ならば $\Delta; \Gamma \vdash^n [N/u]M : A$ である．

【証明】いずれも，項 M の構造に関する帰納法による．ここでは，2 の永続変数の場合のみ示す．
《$M \equiv u$ のとき》(T-PVar) より，$m \leq n$ かつ $A \equiv B$．また，$[N/u]u \equiv N$．前提および補題 2 により，$\Delta; \cdot \vdash^n N : A$．Weakening (補題 3) により結論を得る． □

以上の準備により，subject reduction (定理 1) は，簡約 $M \longrightarrow M'$ の導出に関する帰納法と代入補題により証明される．

4.2 合流性

合流性は，一つの項からの異なる簡約によって生じた異なる項が，それぞれ簡約を繰り返すことによって，再び同一の項へと「合流」させることができることを保証する性質である．厳密には，次のように定式化される．

定理 5 (合流性) 任意の項 M に対して，もし $M \longrightarrow^* M_1$ かつ $M \longrightarrow^* M_2$ ならば，ある項 N が存在して，$M_1 \longrightarrow^* N$ かつ $M_2 \longrightarrow^* N$ である．

ここでは，(1) 標準的な簡約規則 (R-{Beta,Prev,LetBox}) による簡約関係が合流性をもつこと，および，(2) 交換規則 (R-com-{App,Prev,LetBox}) による簡約関係が合流性をもつこと，さらに，(3) 両者の簡約関係が可換であること，を証明する．これらが示されれば，Hindley-Rosen の定理 [1] として知られる，次の性質を用いることにより，交換規則を含む完全な簡約関係もまた合流性をもつことが言える．

定理 6 (Hindley-Rosen) ふたつの簡約関係 $\longrightarrow_1^*$ と $\longrightarrow_2^*$ について，

1. $\longrightarrow_1^*$ と $\longrightarrow_2^*$ がそれぞれ合流性をもち，かつ，
2. $\longrightarrow_1^*$ と $\longrightarrow_2^*$ が可換である，

ならば，$(\longrightarrow_1^* \cup \longrightarrow_2^*)$ も合流性をもつ．

以下，標準的な簡約規則 (R-{Beta,Prev,LetBox}) および合同規則 (R-cong-{Abs,App1,App2,Next,Prev,Box,LetBox1,LetBox2}) で定められる簡約関係を $\longrightarrow_\beta$ により表す．また，交換規則 (R-com-{App,Prev,LetBox}) および合同規則 (R-cong-{Abs,App1,App2,Next,Prev,Box,LetBox1,LetBox2}) で定められる簡約関係を $\longrightarrow_\gamma$ により表す．

まず，$\longrightarrow_\beta$ が合流性を満たすことを示す．

補題 7 ($\longrightarrow_\beta$ は合流性をもつ) 任意の項 M に対して，もし $M \longrightarrow_\beta^* M_1$ かつ $M \longrightarrow_\beta^* M_2$ ならば，ある項 N が存在して，$M_1 \longrightarrow_\beta^* N$ かつ $M_2 \longrightarrow_\beta^* N$ である．

【証明の概略】並列簡約の概念を用いた標準的な手法で行う．すなわち，次の規則 (および合同規則) によって定められる**並列簡約関係** $M \Longrightarrow N$ が diamond property をもつことを，項の構造または並列簡約関係の導出に関する帰納法によって示す．

$$\text{(PR-OVar)}\ x \Longrightarrow x \qquad \text{(PR-PVar)}\ u \Longrightarrow u \qquad \text{(PR-Beta)}\ \frac{M \Longrightarrow M' \qquad N \Longrightarrow N'}{(\lambda x{:}A.\, M)\ N \Longrightarrow [N'/x]M'}$$

$$\text{(PR-Prev)}\ \frac{M \Longrightarrow M'}{\mathsf{prev}\,\mathsf{next}\,M \Longrightarrow M'} \qquad \text{(PR-LetBox)}\ \frac{M \Longrightarrow M' \qquad N \Longrightarrow N'}{\mathsf{let}\ \ \mathsf{box}\,u =_i \mathsf{box}\,M\ \ \mathsf{in}\ N \Longrightarrow [M'/u]N'}$$

ついで，$\Longrightarrow^+$ と $\longrightarrow_\beta^*$ が同等であることを，それぞれの簡約関係の導出に関する帰納法などにより示す．以上より，$\longrightarrow_\beta^*$ も diamond property をもち，したがって $\longrightarrow_\beta$ は合流性をもつ． □

次に，$\longrightarrow_\gamma$ が合流性をもつこと (補題 8)，および，$\longrightarrow_\beta^*$ と $\longrightarrow_\gamma^*$ が可換であること (補題 9) を証明する．

補題 8 ($\longrightarrow_\gamma$ は合流性をもつ) もし $M \longrightarrow_\gamma^* M_1, M \longrightarrow_\gamma^* M_2$ ならば，ある項 M' が存在して，$M_1 \longrightarrow_\gamma^* M'$, $M_2 \longrightarrow_\gamma^* M'$ である．

【証明】$\longrightarrow_\gamma$ が diamond property をもつこと，すなわち $M_1 \longleftarrow_\gamma M \longrightarrow_\gamma M_2$ ならば，
$M_1 \longrightarrow_\gamma M' \longleftarrow_\gamma M_2$ なる項 M' が存在することを，$\longrightarrow_\gamma$ の定義に沿った帰納法により示す． □

補題 9 ($\longrightarrow_\beta^*$ と $\longrightarrow_\gamma^*$ は可換) もし $M \longrightarrow_\beta^* M_\beta, M \longrightarrow_\gamma^* M_\gamma$ ならば，ある項 M' が存在して，$M_\beta \longrightarrow_\gamma^* M'$, $M_\gamma \longrightarrow_\beta^* M'$ である．

【証明】$M_\beta \longleftarrow_\beta M \longrightarrow_\gamma M_\gamma$ ならば，$M_\beta \longrightarrow_\gamma^* M' \longleftarrow_\beta^* M_\gamma$ なる項 M' が存在することを，$\longrightarrow_\beta$ や $\longrightarrow_\gamma$ の定義に沿った帰納法により示す．補題はこれより帰結する． □

以上の準備により，完全な簡約関係 $\longrightarrow$ の合流性は次のように証明される．

【定理 5 の証明】補題 7, 8 より，$\longrightarrow_\beta$ と $\longrightarrow_\gamma$ は，それぞれ合流性をもつ．また，補題 9 より，$\longrightarrow_\beta^*$ と $\longrightarrow_\gamma^*$ は可換である．以上の結果と，Hindley-Rosen の定理 (命題 6) より，$(\longrightarrow_\beta^* \cup \longrightarrow_\gamma^*)$ もまた合流性をもつ．$\longrightarrow^* = (\longrightarrow_\beta^* \cup \longrightarrow_\gamma^*)^*$ であるので，結局 $\longrightarrow$ は合流性をもつ． □

4.3 強正規化性

強正規化性は，型付けされた項に対して，無限長の簡約列が存在しないことを保証する性質である．

定理 10 (強正規化性) $\Delta;\Gamma \vdash^n M : A$ ならば $M \longrightarrow M_1 \longrightarrow \cdots \longrightarrow M_n \longrightarrow \cdots$ なる無限列は存在しない．

ここでは，$\lambda^{\bigcirc\Box}$ の強正規化性を，[5] で述べられている，直和型を導入した型付 λ 計算体系の強正規化性に帰着することによって，間接的に示す．

以下では，まず，直和型を導入した型付 λ 計算 $\lambda^{\vee}$ を定義し，$\lambda^{\bigcirc\Box}$ から $\lambda^{\vee}$ への変換 $|\cdot|$ を定義する．ついで，ここで定義した変換 $|\cdot|$ が型導出と簡約関係を保存することを示し (補題 11, 12)，それによって $\lambda^{\bigcirc\Box}$ の強正規化性が $\lambda^{\vee}$ のそれに帰着できることを示す．

$\lambda^{\vee}$ の構文規則　$\lambda^{\vee}$ の基底型の集合 $\mathbf{BTypes}^{\vee}$，変数の集合 $\mathbf{Vars}^{\vee}$ および構文規則を，次のように定める．

基底型	b	$\in$	$\mathbf{BTypes}^{\vee} = \mathbf{BTypes} \cup \{b^{\circ}\}$	(b° fresh)
変数	z	$\in$	$\mathbf{Vars}^{\vee} = \mathbf{OVars} \cup \mathbf{PVars} \cup \{z^{\circ}\}$	(z° fresh)
型	A, B	$::=$	$b \mid A \to B \mid A + B \mid \bot$	
項	e, f	$::=$	$z \mid \lambda z.\, e \mid e\ f \mid \mathsf{inl}\, e \mid \mathsf{inr}\, e \mid \mathsf{abort}\, e \mid (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2)$	
コンテキスト	Γ	$::=$	$\cdot \mid \Gamma, z{:}A$	

b° は，$\lambda^{\bigcirc\Box}$ の基底型集合 $\mathbf{BTypes}$ に含まれない新たな基底型であり，z° は，$\lambda^{\bigcirc\Box}$ の変数集合 $\mathbf{OVars}$, $\mathbf{PVars}$ に含まれない新たな変数であるとする．これらは，後で定義する変換 $|\cdot|$ において，$\lambda^{\bigcirc\Box}$ の $\bigcirc$ 型や next 式を $\lambda^{\vee}$ で表現するために用いられる．

$\lambda^{\vee}$ の型システム　$\lambda^{\vee}$ の型判断を $\Gamma \vdash^{\vee} e : A$ によって表し，項 e はコンテキスト Γ のもとで型 A をもつことを意味するものとする．また，型付け規則を，次のように定める．

$$(\mathrm{T}^{\vee}\text{-Var})\ \frac{(z{:}A \in \Gamma)}{\Gamma \vdash^{\vee} z : A} \qquad (\mathrm{T}^{\vee}\text{-Abs})\ \frac{\Gamma, z{:}A \vdash^{\vee} e : B}{\Gamma \vdash^{\vee} \lambda z.\, e : A \to B} \qquad (\mathrm{T}^{\vee}\text{-App})\ \frac{\Gamma \vdash^{\vee} e : A \to B \qquad \Gamma \vdash^{\vee} f : A}{\Gamma \vdash^{\vee} e\ f : B}$$

$$(\mathrm{T}^{\vee}\text{-Abort})\ \frac{\Gamma \vdash^{\vee} e : \bot}{\Gamma \vdash^{\vee} \mathsf{abort}\, e : A} \qquad (\mathrm{T}^{\vee}\text{-Inl})\ \frac{\Gamma \vdash^{\vee} e_1 : A_1}{\Gamma \vdash^{\vee} \mathsf{inl}\, e : A_1 + A_2} \qquad (\mathrm{T}^{\vee}\text{-Inr})\ \frac{\Gamma \vdash^{\vee} e : A_2}{\Gamma \vdash^{\vee} \mathsf{inr}\, e : A_1 + A_2}$$

$$(\mathrm{T}^{\vee}\text{-Case})\ \frac{\Gamma \vdash^{\vee} e : A_1 + A_2 \qquad \Gamma, z_1{:}A_1 \vdash^{\vee} f_1 : B \qquad \Gamma, z_2{:}A_2 \vdash^{\vee} f_2 : B}{\Gamma \vdash^{\vee} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2 : B}$$

$\lambda^{\vee}$ の簡約システム　$\lambda^{\vee}$ の簡約関係を $e \xrightarrow{\vee} e'$ によって表し，項 e が e' に 1 ステップで簡約されることを意味するものとする．また，簡約規則を，次のように定める．任意の部分式で簡約を行える full reduction を仮定するが，いわゆる合同規則は省略している．$\xrightarrow{\vee}$ の推移的閉包を $\xrightarrow{\vee}{}^{+}$ と表す．

$(\mathrm{R}^{\vee}\text{-Beta})$	$(\lambda z.\, e)\ f \xrightarrow{\vee} [f/z]e$
$(\mathrm{R}^{\vee}\text{-CaseL})$	$\mathsf{case}\ \mathsf{inl}\, e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2 \xrightarrow{\vee} [e/z_1]f_1$
$(\mathrm{R}^{\vee}\text{-CaseR})$	$\mathsf{case}\ \mathsf{inr}\, e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2 \xrightarrow{\vee} [e/z_2]f_2$
$(\mathrm{R}^{\vee}\text{-com-Abort-App})$	$(\mathsf{abort}\, e)\ f \xrightarrow{\vee} \mathsf{abort}\, e$
$(\mathrm{R}^{\vee}\text{-com-Abort-Abort})$	$\mathsf{abort}(\mathsf{abort}\, e) \xrightarrow{\vee} \mathsf{abort}\, e$
$(\mathrm{R}^{\vee}\text{-com-Abort-Case})$	$(\mathsf{case}\ (\mathsf{abort}\, e)\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2) \xrightarrow{\vee} \mathsf{abort}\, e$
$(\mathrm{R}^{\vee}\text{-com-Case-App})$	$(\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2)\ g \xrightarrow{\vee} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1\ g \mid \mathsf{inr}\, z_2 \Rightarrow f_2\ g \qquad (z_1, z_2 \notin \mathrm{FV}(g))$

($\mathrm{R}^{\vee}$-COM-CASE-ABORT) $\mathsf{abort}(\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2)$
$\xrightarrow{\vee} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow \mathsf{abort}\, f_1 \mid \mathsf{inr}\, z_2 \Rightarrow \mathsf{abort}\, f_2$

($\mathrm{R}^{\vee}$-COM-CASE-CASE) $\mathsf{case}\ (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow f_1 \mid \mathsf{inr}\, z_2 \Rightarrow f_2)\ \mathsf{of}\ \mathsf{inl}\, y_1 \Rightarrow g_1 \mid \mathsf{inr}\, y_2 \Rightarrow g_2$
$\xrightarrow{\vee} \mathsf{case}\ e\ \mathsf{of}\ \mathsf{inl}\, z_1 \Rightarrow\ (\mathsf{case}\ f_1\ \mathsf{of}\ \mathsf{inl}\, y_1 \Rightarrow g_1 \mid \mathsf{inr}\, y_2 \Rightarrow g_2)$
$\mid \mathsf{inr}\, z_2 \Rightarrow\ (\mathsf{case}\ f_2\ \mathsf{of}\ \mathsf{inl}\, y_1 \Rightarrow g_1 \mid \mathsf{inr}\, y_2 \Rightarrow g_2)$
$(z_1, z_2 \notin \mathrm{FV}(g_1) \cup \mathrm{FV}(g_2))$

$\lambda^{\bigcirc\Box}$ から $\lambda^{\vee}$ への変換 $\lambda^{\bigcirc\Box}$ から $\lambda^{\vee}$ への変換 $|\cdot|$ を，次のように定める．

型	$\lvert b\rvert \equiv b$	$\lvert A \to B\rvert \equiv \lvert A\rvert \to \lvert B\rvert$	$\lvert \bigcirc A\rvert \equiv b^\circ \to \lvert A\rvert$	$\lvert \Box A\rvert \equiv \lvert A\rvert + \bot$
項	$\lvert x\rvert \equiv x$	$\lvert u\rvert \equiv u$	$\lvert \lambda x{:}A.\, M\rvert \equiv \lambda x.\, \lvert M\rvert$	$\lvert M\ N\rvert \equiv \lvert M\rvert\, \lvert N\rvert$
	$\lvert \mathsf{next}\, M\rvert \equiv \lambda z^\circ.\, \lvert M\rvert$	$\lvert \mathsf{prev}\, M\rvert \equiv \lvert M\rvert\, z^\circ$	$\lvert \mathsf{box}\, M\rvert \equiv \mathsf{inl}\, \lvert M\rvert$	
	$\lvert \mathsf{let\ box}\, u =_i M\ \mathsf{in}\ N\rvert \equiv \mathsf{case}\ \lvert M\rvert\ \mathsf{of}\ \mathsf{inl}\, u \Rightarrow \lvert N\rvert \mid \mathsf{inr}\, v \Rightarrow \mathsf{abort}\, v$			
コンテキスト	$\lvert\cdot\rvert \equiv \cdot$	$\lvert \Gamma, x{:}^n A\rvert \equiv \lvert\Gamma\rvert, x{:}\lvert A\rvert$	$\lvert \Delta, u{::}^n A\rvert \equiv \lvert\Delta\rvert, u{:}\lvert A\rvert$	
	$\lvert \Delta;\Gamma\rvert^\circ \equiv \lvert\Delta\rvert, \lvert\Gamma\rvert, z^\circ{:}b^\circ$			

このように定義された変換 $|\cdot|$ は，次の 2 つの補題で示される性質をもつ．

補題 11 (変換 $|\cdot|$ は型導出を保存する) もし，$\lambda^{\bigcirc\Box}$ において $\Delta;\Gamma \vdash^n M : A$ ならば，$\lambda^{\vee}$ において $|\Delta;\Gamma|^\circ \vdash^{\vee} |M| : |A|$ である．

【証明】項 M の構造に関する帰納法による．以下では，**next**/**prev**/**box**/**let box** の場合のみ示す．
《$M \equiv \mathsf{next}\, P$ のとき》(T-NEXT) より，ある型 B が存在して $A \equiv \bigcirc B$ かつ $\Delta;\Gamma \vdash^{n+1} P : B$ である．帰納法の仮定より，$|\Delta;\Gamma|^\circ \vdash^{\vee} |P| : |B|$ である．($\mathrm{T}^{\vee}$-ABS) より，$|\Delta|, |\Gamma| \vdash^{\vee} \lambda z^\circ.\, |P| : b^\circ \to |B|$．Weakening により，$|\Delta;\Gamma|^\circ \vdash^{\vee} \lambda z^\circ.\, |P| : b^\circ \to |B|$ すなわち結論を得る．
《$M \equiv \mathsf{prev}\, P$ のとき》(T-PREV) より，時刻 $n-1$ が存在して，$\Delta;\Gamma \vdash^{n-1} P : \bigcirc A$ である．帰納法の仮定より，$|\Delta;\Gamma|^\circ \vdash^{\vee} |P| : b^\circ \to |A|$．また，($\mathrm{T}^{\vee}$-VAR) より $|\Delta;\Gamma|^\circ \vdash^{\vee} z^\circ : b^\circ$ である．($\mathrm{T}^{\vee}$-APP) より
$|\Delta;\Gamma|^\circ \vdash^{\vee} |P|\ z^\circ : |A|$ すなわち結論を得る．
《$M \equiv \mathsf{box}\, P$ のとき》(T-BOX) より，ある型 B が存在して，$A \equiv \Box B$ かつ $\Delta;\cdot \vdash^n P : B$ である．帰納法の仮定より，$|\Delta;\cdot|^\circ \vdash^{\vee} |P| : |B|$ である．($\mathrm{T}^{\vee}$-INL) より，$|\Delta;\cdot|^\circ \vdash^{\vee} \mathsf{inl}\, |P| : |B| + \bot$．さらに weakening により，$|\Delta;\Gamma|^\circ \vdash^{\vee} \mathsf{inl}\, |P| : |B| + \bot$ すなわち結論を得る．
《$M \equiv \mathsf{let\ box}\, u =_i P\ \mathsf{in}\ Q$ のとき》(T-LETBOX) より，ある型 B が存在して，$\Delta;\Gamma \vdash^{n+i} P : \Box B$ かつ $\Delta, u{::}^{n+i} B;\Gamma \vdash^n Q : A$ である．帰納法の仮定より，$|\Delta;\Gamma|^\circ \vdash^{\vee} |P| : |B| + \bot$ かつ $|\Delta;\Gamma|^\circ, u{:}|B| \vdash^{\vee} |Q| : |A|$ である．($\mathrm{T}^{\vee}$-VAR), ($\mathrm{T}^{\vee}$-ABORT) より，$|\Delta;\Gamma|^\circ, v{:}\bot \vdash^{\vee} \mathsf{abort}\, v : |A|$．($\mathrm{T}^{\vee}$-CASE) より，
$|\Delta;\Gamma|^\circ \vdash^{\vee} \mathsf{case}\ |P|\ \mathsf{of}\ \mathsf{inl}\, u \Rightarrow |Q| \mid \mathsf{inr}\, v \Rightarrow \mathsf{abort}\, v$ すなわち結論を得る． □

補題 12 (変換 $|\cdot|$ は簡約関係を保存する) もし，$\lambda^{\bigcirc\Box}$ において $M \longrightarrow M'$ ならば，$\lambda^{\vee}$ において $|M| \xrightarrow{\vee}{}^{+} |M'|$ である．

【証明】$M \longrightarrow M'$ の導出に関する帰納法による．最後に適用した簡約規則により場合分けを行う．以下では，(R-LETBOX) および (R-COM-PREV) の場合のみ示す．
《(R-LETBOX) によるとき，すなわち $M \equiv \mathsf{let\ box}\, u =_i \mathsf{box}\, P\ \mathsf{in}\ Q$, $M' \equiv [P/u]Q$ のとき》
$|M| \equiv \mathsf{case}\ (\mathsf{inl}\, |P|)\ \mathsf{of}\ \mathsf{inl}\, u \Rightarrow |Q| \mid \mathsf{inr}\, v \Rightarrow \mathsf{abort}\, v$, $|M'| \equiv [|P|/u]\,|Q|$ であるので，($\mathrm{R}^{\vee}$-CASEL) より
$|M| \xrightarrow{\vee} |M'|$ である．
《(R-COM-PREV) によるとき，すなわち $M \equiv \mathsf{prev}(\mathsf{let\ box}\, u =_{i+1} P\ \mathsf{in}\ Q)$, $M' \equiv \mathsf{let\ box}\, u =_i P\ \mathsf{in\ prev}\, Q$ のとき》$|M| \equiv (\mathsf{case}\ |P|\ \mathsf{of}\ \mathsf{inl}\, u \Rightarrow |Q| \mid \mathsf{inr}\, v \Rightarrow \mathsf{abort}\, v)\ z^\circ$,

$|M'| \equiv \mathsf{case}\ |P|\ \mathsf{of}\ \mathsf{inl}\, u \Rightarrow |Q|\ z^\circ \mid \mathsf{inr}\, v \Rightarrow \mathsf{abort}\, v$ である．
$e \equiv \mathsf{case}\ |P|\ \mathsf{of}\ \mathsf{inl}\, u \Rightarrow |Q|\ z^\circ \mid \mathsf{inr}\, v \Rightarrow (\mathsf{abort}\, v)\ z^\circ$ とおくと，(R$^\vee$-COM-CASE-APP) より $|M| \xrightarrow{\vee} e$，また (R$^\vee$-COM-ABORT-APP) と合同規則より $e \xrightarrow{\vee} |M'|$ であるので，$|M| \xrightarrow{\vee}{}^+ |M'|$ である． □

以上の準備により，定理 10 は次のように証明される．

【定理 10 の証明】いま，仮に $\lambda^{\bigcirc\Box}$ において，型付け可能であるが強正規化性をもたない項 M が存在するとする．補題 11 より，$\lambda^\vee$ において $|M|$ もまた型付け可能である．$\lambda^{\bigcirc\Box}$ において，M の無限簡約列の 1 つを $M \longrightarrow M_1 \longrightarrow M_2 \longrightarrow \cdots$ とする．すると，補題 12 より，$\lambda^\vee$ においても $|M|$ の無限簡約列
$|M| \xrightarrow{\vee}{}^+ |M_1| \xrightarrow{\vee}{}^+ |M_2| \xrightarrow{\vee}{}^+ \cdots$ が存在することになり矛盾する． □

4.4 時刻順正規化性

時刻順正規化性 (time-ordered normalization) は，直観的には，項の簡約を「時刻順に」行うことができることを示す性質である [3]．これは，プログラムの実行において，ある束縛時の計算の途中で以前の束縛時に遡る必要がないことを保証し，メタプログラミングにおける計算の順序を規定するための指針となるものである．

時刻順正規化性は，Davies によって提唱された概念であり，$\lambda^{\bigcirc\Box}$ の部分体系に相当する $\lambda^\bigcirc$ に対して証明が与えられている [3]．しかし，そこで論じられている時刻順正規化性は，各 β 簡約の間に可能なすべての時相簡約 ($\lambda^{\bigcirc\Box}$ の (R-PREV) 簡約に相当する) を行うという仮定のもとに，β 簡約基のみが時刻順に簡約可能であるという，限定的で煩雑な定義となっていた．また，その定式化および証明はやや非形式的なものであった．

本稿では，3.3 節で既に触れたように，簡約が行われるべき時刻である簡約時刻を明示的にした，簡約関係の定義を行った．これにより，時刻順正規化性は，β 簡約 (R-BETA)，時相簡約 (R-{PREV,LETBOX})，交換規則 (R-COM-{APP,PREV,LETBOX}) を含むすべての簡約が，それらの簡約時刻の順に行えるという，簡潔でより一般的な命題として形式化することが可能となる．以下では，このように一般化された時刻順正規化性を厳密に形式化し，その証明を行う．

定理を厳密に述べるための準備として，まず，ある束縛時以前の計算が完了していることを表す時刻正規形の概念と，時刻中性項 (time neutral term) という，それを変数に代入しても新たな簡約基が形成されることがないような項の概念を導入する．

項 M が，簡約時刻 ℓ 未満の簡約基を含まないとき，M は ℓ **時未満正規 (形)** であるといい，$\Downarrow^\ell M$ によって表す．すなわち，$\Downarrow^\ell M$ であるとは，$M \xrightarrow{\ell'} M'$, $\ell' < \ell$ なる ℓ', M' が存在しないことである．また，項 M が，項時刻 ℓ 未満の λ/**next**/**box**/**let box** 項でないとき，M は ℓ **時未満中性項**であるといい，$\bigtriangledown^\ell M$ によって表す．すなわち，$\bigtriangledown^\ell M$ であるとは，M の項時刻が ℓ 以上であるか，M が x, u, $P\,Q$, $\mathsf{prev}\, P$ のいずれかの形をしていることである．

すると，時刻順正規化性は，次のように定式化される．

定理 13 (時刻順正規化性) もし $\Delta;\Gamma \vdash^n M : A$, $\Downarrow^\ell M$, $M \xrightarrow{\ell} M'$ ならば，$\Downarrow^\ell M'$ である．

この定理を証明するために，より強い性質である time subject reduction (補題 15) を示す．その準備として，まず次の time substitution lemma を証明する．これは，直観的には，代入によって，その代入の時刻よりも過去の簡約基が生成しないという性質である．

補題 14 (Time Substitution Lemma)

1. いま，$\Delta;\Gamma, x{:}^n B \vdash^m M : A$, $\Downarrow^\ell M$ かつ $\Delta;\Gamma \vdash^n N : B$, $\Downarrow^\ell N$ であり，$\ell \le n$, $M' \equiv [N/x]M$ であるとする．このとき，$\Downarrow^\ell M'$ であり，また，$\bigtriangledown^\ell M$ ならば $\bigtriangledown^\ell M'$ である．

2. いま，$\Delta, u{::}^{n}B; \Gamma \vdash^{m} M : A$, $\Downarrow^{\ell} M$ かつ $\Delta; \cdot \vdash^{n} N : B$, $\Downarrow^{\ell} N$ であり，$\ell \leq n$, $M' \equiv [N/u]M$ であるとする．このとき，$\Downarrow^{\ell} M'$ であり，また，$\bigtriangledown^{\ell} M$ ならば $\bigtriangledown^{\ell} M'$ である．

【証明】いずれも，項 M の構造に関する帰納法による．以下では，$P' \equiv [N/x]P$, $Q' \equiv [N/x]Q$ とし，1 の通常変数/関数抽象/関数適用/**let box** の場合のみ示す．
《$M \equiv x$, $M' \equiv N$ のとき》$\Downarrow^{\ell} N$ より，すなわち $\Downarrow^{\ell} M'$．また M' の時刻は n なので，$\ell \leq n$ より $\bigtriangledown^{\ell} M'$．
《$M \equiv y \neq x$, $M' \equiv y$ のとき》$M \equiv M'$ より，明らか．
《$M \equiv \lambda y{:}C.\, P$, $M' \equiv \lambda y{:}C.\, P'$ のとき》$x \neq y$ と仮定してよい．(T-Abs) より，ある型 D が存在して，$\Delta; \Gamma, x{:}^{n}B, y{:}^{m}C \vdash^{m} P : D$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$．帰納法の仮定より，$\Downarrow^{\ell} P'$．ゆえに，$\Downarrow^{\ell}(\lambda y{:}C.\, P')$ すなわち $\Downarrow^{\ell} M'$．また，M と M' の時刻が等しいので，$\bigtriangledown^{\ell} M$ ならば $\bigtriangledown^{\ell} M'$．
《$M \equiv P\ Q$, $M' \equiv P'\ Q'$ のとき》(T-App) より，ある型 C が存在して，$\Delta; \Gamma, x{:}^{n}B \vdash^{m} P : C \to A$ かつ $\Delta; \Gamma, x{:}^{n}B \vdash^{m} Q : C$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$ かつ $\Downarrow^{\ell} Q$ かつ $\bigtriangledown^{\ell} P$．帰納法の仮定より，$\Downarrow^{\ell} P'$ かつ $\Downarrow^{\ell} Q'$ かつ $\bigtriangledown^{\ell} P'$．ゆえに，$\Downarrow^{\ell}(P'\ Q')$ すなわち $\Downarrow^{\ell} M'$．また，M' の構造から $\bigtriangledown^{\ell} M'$．
《$M \equiv$ **let box** $v =_i P$ **in** Q, $M' \equiv$ **let box** $v =_i P'$ **in** Q' のとき》(T-LetBox) より，ある型 C が存在して，$\Delta; \Gamma, x{:}^{n}B \vdash^{m+i} P : \Box C$ かつ $\Delta, v{::}^{m+i}C; \Gamma, x{:}^{n}B \vdash^{m} Q : A$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$ かつ $\Downarrow^{\ell} Q$ であり，$\bigtriangledown^{\ell} P$ または $\ell \leq m + i$．帰納法の仮定より，$\Downarrow^{\ell} P'$ かつ $\Downarrow^{\ell} Q'$ であり，$\bigtriangledown^{\ell} P'$ または $\ell \leq m + i$．ゆえに，$\Downarrow^{\ell}$(**let box** $v =_i P'$ **in** Q') すなわち $\Downarrow^{\ell} M'$．また，M と M' の時刻が等しいので，$\bigtriangledown^{\ell} M$ ならば $\bigtriangledown^{\ell} M'$．□

次に，簡約が，項の時刻正規性と時刻中性を保存するという time subject reduction を証明する．この補題は，時刻順正規化性 (定理 13) を含む命題となっている．

補題 15 (Time Subject Reduction) いま，$\Delta; \Gamma \vdash^{n} M : A$, $\Downarrow^{\ell} M$, $M \xrightarrow{\ell} M'$ であるとする．このとき，$\Downarrow^{\ell} M'$ であり，また，$\bigtriangledown^{\ell} M$ ならば $\bigtriangledown^{\ell} M'$ である．

【証明】簡約 $M \xrightarrow{\ell} M'$ の導出に関する帰納法による．最後に適用した簡約規則により場合分けを行う．以下では，(R-Beta), (R-Prev), (R-LetBox), (R-com-App) の場合のみ示す．
《(T-Beta) により，$M \equiv (\lambda x{:}B.\, P)\ Q$, $M' \equiv [Q/x]P$ のとき》(T-{App,Abs}) より，$\Delta; \Gamma, x{:}^{n}B \vdash^{n} p : A$ かつ $\Delta; \Gamma \vdash^{n} Q : B$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$ かつ $\Downarrow^{\ell} Q$．$M \xrightarrow{\ell} M'$ より，$\ell = n$ すなわち $\ell \leq n$．Time substitution lemma (補題 14) より，$\Downarrow^{\ell} M'$．また，M' の時刻は n なので，$\ell = n$ より $\bigtriangledown^{\ell} M'$．
《(T-Prev) により，$M \equiv$ **prev next** P, $M' \equiv P$ のとき》(T-{Prev,Next}) より，時刻 $n-1$ が存在して，$\Delta; \Gamma \vdash^{n} P : A$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$．$M \xrightarrow{\ell} M'$ より，$\ell = n - 1$．$\Downarrow^{\ell} P$ より，すなわち $\Downarrow^{\ell} M'$．また，M' の時刻は n なので，$\ell + 1 = n$ より，$\bigtriangledown^{\ell} M'$．
《(T-LetBox) により，$M \equiv$ **let box** $u =_i$ **box** P **in** Q, $M' \equiv [P/u]Q$ のとき》(T-{LetBox,Box}) より，ある型 B が存在して，$\Delta; \cdot \vdash^{n+i} P : B$ かつ $\Delta, u{::}^{n+i}B \vdash^{n} Q : A$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$ かつ $\Downarrow^{\ell} Q$．$M \xrightarrow{\ell} M'$ より，$\ell = n + i$ すなわち $\ell \leq n + i$．Time substitution lemma (補題 14) より，$\Downarrow^{\ell} M'$．また，$\bigtriangledown^{\ell} M$ ならば，$\ell \leq n$ であり，M' の時刻が n であるので，$\bigtriangledown^{\ell} M'$．
《(T-com-App) により，$M \equiv$ (**let box** $u =_i P$ **in** Q) R, $M' \equiv$ **let box** $u =_i P$ **in** $(Q\ R)$ のとき》(T-{App, LetBox}) より，ある型 B, C が存在して，$\Delta; \Gamma \vdash^{n+i} P : \Box C$ かつ $\Delta, u{::}^{n+i}C; \Gamma \vdash^{n} Q : B \to A$ かつ $\Delta; \Gamma \vdash^{n} R : B$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$ かつ $\Downarrow^{\ell} Q$ かつ $\Downarrow^{\ell} R$．$M \xrightarrow{\ell} M'$ より，$\ell = n$．Q は λ/**let box** であり得るが，Q の項時刻が n なので，$\ell = n$ より $\Downarrow^{\ell}(Q\ R)$ であり，また P の項時刻が $n + i$ なので，結局 $\Downarrow^{\ell} M'$．また，M' の項時刻が n なので，$\ell = n$ より $\bigtriangledown^{\ell} M'$．
《合同規則により，$M \equiv P\ Q$, $M' \equiv P'\ Q$, $P \xrightarrow{\ell} P'$ のとき》(T-App) より，ある型 B が存在して，$\Delta; \Gamma \vdash^{n} P : B \to A$ かつ $\Delta; \Gamma \vdash^{n} Q : B$．$\Downarrow^{\ell} M$ より，$\Downarrow^{\ell} P$ かつ $\Downarrow^{\ell} Q$ かつ $\bigtriangledown^{\ell} P$．帰納法の仮定より，$\Downarrow^{\ell} P'$ かつ $\bigtriangledown^{\ell} P'$．ゆえに，$\Downarrow^{\ell}(P'\ Q)$ すなわち $\Downarrow^{\ell} M'$．また，M' の構造から $\bigtriangledown^{\ell} M'$．□

5 関連研究

様相論理に基づく計算体系 Davies による $\lambda^{\bigcirc}$ [3] は，オフライン部分計算のための束縛時解析 (binding-time analysis) のための型システムを備えた計算体系である．Curry-Howard の同型対応を (□ を含まない) 線形時間時相論理に拡張し，$\bigcirc A$ という論理式を，次の束縛時で実行可能なコードの型に対応づけている．$\lambda^{\bigcirc}$ におけるコードは，内部に対象レベルの変数の自由な出現を許容することから，表現力に富み，効率的なコードの生成のための部分計算を実現できる．一方で，コードの汎用性の点では制限があり，束縛時をまたがって利用可能なコードを記述することはできず，コードを即時実行するための機構は用意されていない．

Davies と Pfenning は，Curry-Howard の同型対応を S4 様相論理に拡張し，$\Box A$ という論理式を，自由変数を持たないコードの型と解釈することにより，実行時コード生成の機構を持つ計算体系 $\lambda^{\Box}$ [4] を構築した．$\lambda^{\bigcirc}$ と異なり動的コード生成機構を備えているが，一方で，扱うコードは自由変数を含むことができないために，$\lambda^{\bigcirc}$ が実現したような効率のよいコード生成は行えない (3.4 節の例参照)．

本稿で導入した $\lambda^{\bigcirc\Box}$ は，互いに相補的な体系である $\lambda^{\bigcirc}$ と $\lambda^{\Box}$ を統合するものであり，Curry-Howard の同型対応を □ を含む線形時間時相論理に拡張し、様相 ○ と □ をともに整合的に扱う体系となっている．

Miyamoto と Igarashi による $\lambda_s^{\Box}$ [7] は，セキュリティのための情報流解析 (information flow analysis) の型システムを備えた計算体系である．Curry-Howard の同型対応を local validity という様相をもつ論理に拡張し，可能世界 ℓ から (0 ステップ以上で) 到達できる任意の世界で A が成立する，という意味の $\Box_\ell A$ という論理式を，セキュリティレベル ℓ 以上でアクセス可能な式の型に対応づけている．$\lambda_s^{\Box}$ の型 $\Box_\ell A$ と $\lambda^{\bigcirc\Box}$ の型 $\bigcirc^i \Box A$ は，前者では絶対的なレベルの指定，後者では相対的なレベルの指定であるという違いはあるものの，いずれも特定のレベル以降で利用可能な型の表現である点で，類似している．$\lambda_s^{\Box}$ の主要な性質の一つである noninterference は，高レベルの計算結果が低レベルに計算に影響しないこと，言いかえれば，低レベルの計算を行うために高レベルの計算が必要ないことを示すものであるが，これは，本稿で証明した $\lambda^{\bigcirc\Box}$ の時刻順正規化性とも，深い関連が予想される．

メタプログラミングのための型システム $\lambda^{\bigcirc}$ と $\lambda^{\Box}$ におけるコード型は，コード内の自由変数の出現の可否に着目して，それぞれ「開いたコード型」,「閉じたコード型」と対比されることがある．この立場から，これらの 2 種のコード型を統一的に扱う計算体系の研究が，近年行われてきている．これらの研究の多くは， $\lambda^{\bigcirc}$ あるいは $\lambda^{\Box}$ の一方を基盤とし，その表現力を洗練・拡張するという手法をとっている．

Taha と Sheard による MetaML [11] は，$\lambda^{\bigcirc}$ を拡張した体系で，$\bigcirc A$ に相当する開いたコード型 $\langle A \rangle$ 型をもち，コードの実行のための **run** 構文を備えている．Moggi らの AIM (An Idealized MetaML) [8] は，MetaML に，閉じたコード型である $[A]$ を導入し，より単純な型システムでコードの安全な即時実行を可能にした．また，Benaisssa らによる λ^{BN} [2] は，AIM における $[A]$ 型の意味を，閉じた (コードに限らない) 値の型と解釈することで，体系の整理を行っている．すなわち，AIM では独立した型と扱われていた「閉じたコード」を，λ^{BN} では $[\langle A \rangle]$ 型を持つ特殊な「開いたコード」として扱う．

Nanevski は，$\lambda^{\Box}$ を基盤として，名前 (name) と呼ばれる，Lisp のシンボルに近い概念を導入した $\nu^{\Box}$ [9] を提案している．$\nu^{\Box}$ は，**box** コード式に名前の自由な出現を許し，また **box** 内部において名前によるシンボリックな計算を行えるよう拡張し，3.4 節でみた $\texttt{power}^{\bigcirc}$ 関数と同様に，効率の良いコードの生成を可能にしている．

これらの体系はいずれも，$\lambda^{\bigcirc}$ もしくは $\lambda^{\Box}$ を拡張することによって，より汎用的なメタプログラミングを実現しようとするものである．その目標は本稿と共通のものであるが，拡張のアプローチは実用主義的な性格が強く，論理に基づくという本稿の立場とは異なっている．

また，Lisp における擬似引用機構や超変数といった概念を表現するために，山本ら [14] は λq という型付計算体系を提案している．この型システムは本質的には $\lambda^{\bigcirc}$ と同じものであるが，$\lambda^{\bigcirc}$ や $\lambda^{\bigcirc\Box}$ とは異なり，引用

(コードに対応する) 内の超変数への代入時に変数の動的束縛が発生するような体系になっている.

6 おわりに

本稿で我々は，計算体系 $\lambda^{\bigcirc\Box}$ を導入し，その基本的な性質を述べた．$\lambda^{\bigcirc\Box}$ は Curry-Howard の同型対応の拡張により必然様相を導入した線形時間時相論理に基づいており，先行研究の $\lambda^{\bigcirc}$ および $\lambda^{\Box}$ を含む体系となっている．また，$\lambda^{\bigcirc\Box}$ は subject reduction, 合流性，強正規化性，時刻順正規化性の各性質を満たす．時刻順正規化性に関しては，commuting conversion を導入した簡約システムにより成立することを示し，また，厳密な定式化と明快な証明を与えた.

今後の課題としては，時刻順正規化性を反映した評価システムを完成させることが挙げられる．時刻順に決定的に評価を行う戦略を定義することで，実際のプログラミング言語へ適用する基盤となると考えている．また，線型時間時相論理の証明体系として考えた時に，知られている公理系に対して健全/完全になるように拡張すること，それがメタプログラミングとしてどういった意味があるか考察することも興味深い課題である.

参考文献

[1] H. P. Barendregt. *The Lambda Calculus*. North Holland, Amsterdam, The Netherlands, revised edition, 1984.

[2] Zine El-Abidine Benaissa, Eugenio Moggi, Walid Taha, and Tim Sheard. Logical modalities and multi-stage programming. In *Proceedings of Workshop on Intuitionistic Modal Logics and Applications (IMLA'99)*, Trento, Italy, July 1999. Available from `http://www.disi.unige.it/person/MoggiE/publications.html`.

[3] Rowan Davies. A temporal-logic approach to binding-time analysis. In *Proceedings of 11th Annual Symposium on Logic in Computer Science (LICS'96)*, pp. 184–195, 1996.

[4] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *Proceedings of the 23rd Annual ACM Symposium of Principles of Programming Languages (POPL'96)*, pp. 258–270, 1996.

[5] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[6] Robert Glück and Jesper Jørgensen. Efficient multi-level generating extensions for program specialization. In *Proceedings of Programming Languages, Implementations, Logics and Programs (PLILP'95)*, LNCS 982, pp. 259–278, 1995.

[7] Kenji Miyamoto and Atsushi Igarashi. A modal foundation for secure information flow. In *Proceedings of Workshop on Foundations of Computer Security (FCS'04)*, pp. 187–203, 2004.

[8] Eugenio Moggi, Walid Taha, Zine El-Abidine Benaissa, and Tim Sheard. An idealized MetaML: Simpler, and more expressive. In *Proceedings of European Symposium on Programming*, pp. 193–207, 1999.

[9] Aleksandar Nanevski. Meta-programming with names and necessity. In *Proceedings of the seventh ACM SIGPLAN International Conference on Functional Programming (ICFP'02)*, pp. 206–217, 2002. See also `http://www-2.cs.cmu.edu/~aleks/papers/necessity/techrep2.ps`.

[10] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, Vol. 11, No. 4, pp. 511–540, 2001.

[11] Walid Taha and Tim Sheard. Multi-stage programming with explicit annotations. In *Proceedings of Partial Evaluation and Semantics-Based Program Manipulation (PEPM'97)*, pp. 203–217, Amsterdam, The Netherlands, June 1997. ACM Press.

[12] Philip Wickline, Peter Lee, and Frank Pfenning. Run-time code generation and Modal-ML. In *Proceedings of SIGPLAN Conference on Programming Language Design and Implementation (PLDI'98)*, pp. 224–235, 1998.

[13] 湯瀬芳洋. メタプログラミングのための時相論理に基づく型付 λ 計算. Master's thesis, 京都大学大学院情報学研究科知能情報学専攻, February 2005.

[14] 山本和樹, 岡本暁広, 五十嵐淳, 佐藤雅彦. 擬似引用を持つ型付計算体系 λq. 第 5 回プログラミングおよびプログラミング言語ワークショッ プ (PPL2003) 論文集, pp. 87–102, 富士市, March 2003. 日本ソフトウェア科学会. `http://wwwfun.kurims.kyoto-u.ac.jp/~nisimura/ppl2003/`.