HOLを用いた代入定理の検証

小合 敬之

博士 (情報学)

総合研究大学院大学
複合科学研究科
情報学専攻

平成23年度
(2011)

概要

本論文では，高階論理証明システム HOL を用いて代入定理を形式化し検証する．代入定理とは，与えられたラムダ項と与えられたいくつかのその変数に対して，任意の弱正規化可能な同じラムダ項を代入してもその項が弱正規化可能であるなら，任意の弱正規化可能な異なるラムダ項を代入してもその項が弱正規化可能であるという 2006 年に証明された型なしラムダ計算の新しい定理である．これは，原論文では弱正規化可能性に関する代入定理とよばれている．我々は原論文の数学的内容を忠実に形式化する．束縛変数の名前換えの無いラムダ項を扱うために，Homeier のパッケージによって文脈 α-同値を使用する．本検証の結果，ラムダ計算の新しく重要な定理を検証することができた．

本論文の第 1 章では，序論として代入定理の検証の概要とその意義，この検証が成す貢献，関連研究，本論文の構成について説明する．先行研究において採られたラムダ計算の形式化の手法について論じる．また，ラムダ計算の定理を検証した先行研究では Church-Rosser の定理など良く知られた定理が検証されているのに対して，本論文では最近証明されたラムダ計算の新しく重要な定理を検証したことを述べる．第 2 章では，予備知識として型無しラムダ計算と高階論理の自動証明システムである HOL について説明する．HOL については，その論理体系と検証コードの文法，定理の検証で用いる goal，tactic，tactical について説明する．第 3 章では，予備知識として代入定理の数学的記述とその数学的な証明を説明する．代入定理の記述を与え，この定理の証明に導入されたコントロールパスと隣接コントロールパスの概念を導入する．代入定理を証明するための主補題を示し，それを用いて代入定理の証明を与える．第 4 章では，予備知識として文脈 α-同値と二種類のラムダ計算について説明する．pure λ-calculus は pre-λ-calculus の α-同値関係による同値類として定義される．pre-λ-calculus，文脈 α-同値と α-同値，pure λ-calculus を順に定義し，λ-calculus において pure λ-calculus と α-同値を導入することの利点と問題点について述べる．また，ラムダ項の代入の定義に

ついて説明する．第 5 章では，第 3 章と第 4 章の内容を形式化し，代入定理の形式化を与える．検証の目標となる代入定理の論理式の形式化を与え，弱正規化可能性など検証に必要なラムダ計算に関する概念の形式化を与える．部分項の概念を形式化するために，部分項の indicator を導入する．コントロールパス，隣接変数出現，隣接コントロールパスなど代入定理の証明のために導入された概念の形式化を与える．特にラムダ項 M 中の集合 S からの隣接コントロールパスがあることの否定について新たな述語を与えて形式化する．第 6 章では，代入定理の形式的な証明が得られることを示す．ラムダ計算の概念を形式化した関数や述語について，代入定理の証明で必要な補題を形式化して証明する．また，indicator による部分項の表示やコントロールパスなどの代入定理の証明に導入された概念の性質についての補題を形式化して証明する．次に第 3 章で述べた代入定理の証明に用いた補題を形式化して証明するが，円滑な証明をするために，主補題を変更する．そして代入定理の形式的証明の実行について示し，その実行結果について述べて，コード量および処理時間を記す．第 7 章では，この研究の解決した問題を論じる．問題解決の第一は，変数の捕捉による困難を解決するために，束縛変数の名前の集合 S に対し，PWN_S という概念を導入したことである．これは，原論文で使用される永続的弱正規化を置き換え，あるラムダ項がこの性質をもつことを示すときに，β-簡約により起こる代入処理を単純化する．第二に，変数代入によりコントロールパスが保たれることを検証するために帰納法に適切な変数を選択する工夫を用いた．この性質の形式的な記述は両方のスタイルのラムダ項を混在させて用いている．我々は記述中に隠された内側の変数を最外部に移動して帰納法を適用する変数として使用する．第三に，隣接コントロールパスの存在の否定を表現するために新たな述語を導入した．隣接コントロールパスの存在の忠実な形式化は暗黙の自由変数条件を正しく表さない．このため，命題の仮定でその否定を使用する場合は，形式的証明が進まなくなる．この問題を解決するために，我々はその否定が自由変数条件を正しく表すような新たな述語を導入する．そして第 8 章では本研究についてのまとめと今後の研究課題について述べる．

Abstract

In this thesis, we formalizes and verifies Substitution Theorem by using the higher-order theorem prover HOL. Substitution Theorem for weakly normalisation is a new theorem in untyped lambda calculus, which was proved in 2006. This theorem states that for a given lambda term and given free variables in it, the term becomes weakly normalizing when we substitute arbitrary weakly normalizing terms for these free variables, if the term becomes weakly normalizing when we substitute a single arbitrary weakly normalizing term for these free variables. We faithfully formalize mathematical content in the original paper which proved the theorem. To handle λ-terms without renaming bound variables, we use the contextual α-equivalence with the package in Homeier's paper. We formalized and verified the new and significant theorem in untyped lambda calculus.

Section 1 is an introduction. We explain the verification of Substitution Theorem and its achievements. We also discuss about the difference between this thesis and related preceding studies in lambda calculus. Section 2 explains about untyped lambda calculus and HOL. Section 3 describes the mathematical statement of Substitution Theorem and its mathematical proof. We introduce the concepts of a control path and adjacent control paths. Section 4 explains the contextual α-equivalence. Section 5 gives our formalization of the theorem and fomalize a control path, adjacent variable occurrences and adjacent control paths. Section 6 discusses formal proofs of the theorem and lemmas. Section 7 explains our challenges in this work. They are an introduction parametrized persistent weak normalization, a proof of preservation of control paths by substitution and a definition of new pedicate which expresses a negation of existence of adjacent control paths. Section 8 concludes.

目 次

第1章　序論

1.1　概要

自動化された定理証明は盛んに研究されている．数学者によって証明されたとされる定理であっても，証明に誤りがある可能性は考えられる．HOL[19] などの定理証明システムを用いた定理の形式的検証は，論理的推論のレベルに戻って証明を機械的にチェック出来るので，定理が正しいことが検証により確実になる．このように定理検証には数学に貢献する意義があり，さらに四色定理など証明の難しい定理を検証しようという研究も活発である．例えば，Coq[18] による四色定理の検証は歴史的に長い間未解決問題であった難問の長大な証明を検証したものであり，驚くべき結果であった [4]．HOL，Coq や Agda[17] のような定理証明システムは精力的に開発されている．さらにはハードウェアやソフトウェアの検証への応用も広く議論されている．

ラムダ計算の定理の検証も積極的に研究されている．Church-Rosser の定理は，[11] によって検証された．Pure Type System におけるいくつかの性質は [10] によって形式化および検証された．ラムダ計算の α-変換をより上手く形式化するために，文脈 α-同値が [7] によって提案された．よりよいラムダ計算の形式化のために [15] はノミナル論理を導入し，Church-Rosser の定理を含むいくつかの性質を検証した．

本論文では，高階論理証明システム HOL を用いて代入定理を形式化し検証する．これは，与えられたラムダ項と与えられたいくつかのその変数に対して，任意の弱正規化可能な同じラムダ項を代入してもその項が弱正規化可能であるなら，任意の弱正規化可能な異なるラムダ項を代入してもその項が弱正規化可能であるという 2006 年に証明された型なしラムダ計算の新しい定理である [14]．これは，原論文では弱正規化可能性に関する代入定理とよばれているものである．本論文では，これを単に代入定理とよぶ．定理の記述は簡単だが，その証明は，記述の中に隠れている複雑なテクニックを使用している．代入定理は，[14] で与えられる奥深い結果全体の一部である．この

定理は，新しく重要であるため，検証する意義がある．すなわち，本研究の定理検証の意義の数学的側面は，ラムダ計算の新しく重要な定理を検証することである．機械的チェックによって論理的推論のレベルに戻って検証されれば，代入定理が誤りなく証明された定理であることがより確実になる．また，形式化の途上で，証明の簡単化や，新しい概念，新しい定理などを発見する可能性がある．本研究では，結果的には，誤りの訂正として，隣接コントロールパスの自由変数の条件を正しくした定義，新しい概念として，パラメータ付き永続弱正規化可能性が得られる．

HOL は型推論や goal 指向の証明手段を持つ高階論理の証明システムで，様々な理論の形式化や検証に用いられている．Coq など他の定理検証系と同じく，代入定理の検証に十分な能力を備えている．ラムダ計算においても，[7] 等の先行研究があって，ラムダ計算の新しい定理である代入定理の検証にとって良好な環境にあるシステムと言える．我々は [14] の数学的内容を忠実に形式化する．束縛変数の名前換えの無いラムダ項を扱うために，[7] のパッケージによって文脈 α-同値を使用する．検証技術上の意義としては，本研究は，[7] のパッケージが，Church-Rosser の定理以外の大きな定理に適用できることを示すことがである．また，束縛変数名を用いるラムダ計算の定理を検証し，pre-λ calculus に対する束縛変数に関する多数の補助補題を証明し，ラムダ計算の検証の蓄積を増すものである．さらに，コントロールパスの変数代入による保存を示す部分は，検証の際，帰納法の変数をどのように選ぶかを示したケーススタディの形になっている．

束縛変数の名前換えの無いラムダ項の性質を検証した研究結果がいくつかある [11, 10, 7, 15]．これらは α-同値を扱う新しい方法を開発し，それらを束縛変数の名前換えの無いラムダ項のいくつかの性質の検証に適用した．しかし，それらのほとんどは Church-Rosser の定理のようなよく知られた定理のみを検証している．一方，我々は束縛変数の名前換えの無いラムダ項を用いて新しく重要な定理を検証する．我々の検証はまた，[7] で与えられた文脈 α-同値のアイデアとそれに基づく HOL のパッケージが束縛変数の名前換えの無いラムダ項の有力な形式化であることを示している．[7] ではこのアイデアは Church-Rosser の定理の検証に用いられた．しかし，その他の応用結果はまだ示されていない．我々は彼の HOL のパッケージを新しく重要な定理の検証に用いて，彼のアイデアが有効であることを示す．

1.2 本論文の貢献

我々の検証には 5 点の成果がある．

第一に，我々は新しく重要な定理を証明する．型無しラムダ計算はよく研究され，新しい定理を得ることは困難であるので，この分野の新しい定理は，通常の直観力では気付くことのできない意外な概念の細やかな取り扱いを必要とする難解な定理となることが多い．重要な定理は大抵困難な証明が必要となる．実際代入定理の証明は複雑で，隣接コントロールパスという意外な概念の細やかな取り扱いを必要し，束縛変数列を追跡する難しい技法を使用する．

第二に，束縛変数の名前換えのあるラムダ項だけでなく，束縛変数の名前換えの無いラムダ項も必要とする証明を検証する．コントロールパスは代入定理の証明の鍵となる概念であり，束縛変数の名前を使用して定義されている．したがって，我々の形式化は，束縛変数の名前換えの無いラムダ項を扱う必要がある．

第三に，変数の捕捉による困難を解決するために，束縛変数の名前の集合 S に対し，PWN_S という概念を導入する．これは，[14] で使用される永続的弱正規化を置き換え，あるラムダ項がこの性質をもつことを示すときに，β-簡約により起こる代入処理を単純化する．この概念がなければ，我々の形式的証明ははるかに複雑になるであろう．

第四に，変数代入によりコントロールパスが保たれることを検証するために帰納法に適切な変数を選択する工夫を用いる．この性質は [14] で証明することなく暗黙的に使用されたが，その形式的な証明は難しい．なぜならコントロールパスは，束縛変数の名前換えの無いラムダ項に定義された概念である一方，代入は束縛変数の名前換えのあるラムダ項に定義された概念だからである．それゆえこの性質の形式的な記述は両方のスタイルのラムダ項を混在させて用いている．この困難を解決するために，我々は記述中に隠された内側の変数を最外部に移動して帰納法を適用する変数として使用する．

第五に，隣接コントロールパスの存在の否定を表現するために新たな述語を導入する．隣接コントロールパスの存在の忠実な形式化は暗黙の自由変数条件を正しく表さない．このため，命題の仮定でその否定を使用する場合は，形式的証明が進まなくなる．この問題を解決するために，我々はその否定が自由変数条件を正しく表すような新たな述語を導入する．

この検証では，10119 行の HOL のコードと 326 の補題を用いる．HOL のソースコードは `http://research.nii.ac.jp/~tatsuta/hol` で公開している．

1.3 関連研究

定理証明システムによって難しい定理，重要な定理を形式化して証明，検証する研究は多方面にわたる．[4] は Coq を用いて四色定理の検証を成功させた．四色定理とは，平面上の領域を隣接した領域が同じ色にならないように，四色だけで色分けできるという定理である．長年未解決問題だった四色定理を，定理証明システムによって検証できたというのは驚くべき結果であった．hypermap や cycle-counting，planarity，reducibility，unavoidability などの概念を Coq 上に形式化して，新しい性質が検証された．ラムダ計算の定理の証明も積極的に検証されている．Church-Rosser の定理は，[11] によって検証された．Pure Type System におけるいくつかの性質は [10] によって形式化および検証された．これらの研究と比べて，本論文ではラムダ計算の新しい定理である代入定理を検証した点が異なる．

α-同値の定義や形式化は様々な研究が成されている．[2] は pre-λ-term に対して，代入を total に定め，代入を用いて，pre-λ-term に対して α-簡約を定めた．α-同値から pure λ-term を定義することはしていないが，暗黙に，全ての証明は α-同値類に対してなされている．この本はラムダ計算の基礎を与え，本論文の基礎を与える．この本は α-同値に関する考察が不足しておりそのままでは形式化できないが，本論文では，文脈 α-同値のアイデアを導入した [7] による α-同値の形式化を採用している点が異なる．また，この本は形式化は論じていない．

[6] は pre-λ-term に対して，代入を total に定め，代入を用いて，pre-λ-term に対して α-簡約を定めた．α-同値を解析するために，制限された α-同値 $\rhd_{\alpha 0}$ を導入した．これは，$\lambda x.e \rhd_{\alpha_0} \lambda y.e'$ を $\lambda x.e \to_\alpha \lambda y.e'$ かつ $x, y \notin BV(M)$ と定めるものである．制限された α-同値を用いて α-同値を解析した．[6] の制限された α-同値に替えて，本論文では文脈 α-同値を用いて α-同値を定義した [7] による形式化を採用した点が異なる．また，この本は形式化は論じていない．

[16] は pre-λ-term に対して，代入を total だが部分的にのみ正しい形に定

めた．この代入では，$(\lambda y.e')[x := e]$ が $x \neq y$ であっても $y \in FV(e)$ のときは $\lambda y.e'$ と定義する．この定義は普通の代入の意味では正しくないが，これにより代入を total に定義することができる．代入を用いて，pre-λ-term に対して制限された α-簡約を定めた．この制限された α-簡約は, 普通の α-簡約の上に条件として，e 中に $x \in FV(e')$ となる部分項 $\lambda y.e'$ がないという条件のとき，$\lambda x.e \rightarrow_\alpha \lambda y.e[x := y]$ と定義される．[16] は普通の α-同値とこのこの制限された α-簡約から導出される同値が同等であることを証明した．またこれを用いて $\rightarrow_\alpha^*$ のあとで並行簡約を行う一連の簡約のダイアモンド性を形式化し，Isabelle/HOL で検証した．ここで並行簡約は普通の Tait, Martin-Lof, Takahashi の parallel reduction のことであり，ダイアモンド性とは，1-step が合流性をもつことである．検証では Barendregt Variable Convention を用いている．彼らの制限された α-同値に替えて，本論文では文脈 α-同値を用いてアルファ同値を定義した [7] による形式化を採用した点が異なる．彼らは，並行簡約のダイアモンド性を検証したが，本論文では，代入定理を検証した点が異なる．Barendregt convention も用いている点は同じである．

[5] は α-変換を持つラムダ計算を定義付ける五つの公理を導入し，well-formed de Bruijn term を用いてその健全性を示し，これらの公理からいくつかの定理を証明した．またこれらを HOL で形式化し検証した．公理は，(1) ラムダ項の自由変数を返す関数 Fv がある．(2) 変数条件の付いた代入がある．(3) α-変換がある．(4) ラムダ項の構造に基づいて定義された関数は一意に存在する．(5) ABS は変数を取ってラムダ項を返す関数を引数とし，lambda abstraction を返す関数である．の五つである．[9] は [5] の公理を用いて，HOL 上で正規化定理などの検証を行った．本論文では，本論文では文脈 α-同値を用いてアルファ同値を定義した [7] による形式化を採用した点が異なる．また代入定理を検証した点が異なる．

ラムダ計算の α-変換をより上手く形式化するために，文脈 α-同値が [7] によって提案された．これは代入を用いずに α-同値を定義する方法で，代入を用いて定義する方法に比べ定義の明快さを目指したものといえる．[7] は二種類のラムダ計算と文脈 α-同値を与えた．二種類のラムダ計算の一つは pre-λ-calculus で，もう一つは pure λ-calculus である．pre-λ-calculus は束縛変数の名前換えの無いラムダ項を扱う．つまり，$\lambda xy.x \neq \lambda uy.u \neq \lambda uv.u$ となる．また，$x_1 = x_2$ かつ $M_1 = M_2$ のとき，そのときに限り $\lambda x_1.M_1 = \lambda x_2.M_2$ が成り立つ．文脈 α-同値は α-同値関係を定義するために pre-λ-

calculus 上に定義される．関係 $t_1\ ^{xs}\equiv_\alpha^{ys}\ t_2$ は，リスト xs にある各変数をリスト ys にある対応する変数で置き換えるとき，t_1 は t_2 と α-同値であることを意味する．pure λ-calculus は pre-λ-calculus の α-同値関係による同値類 $\Lambda_1/\equiv_\alpha$ である．pre-λ-calculus は他の束縛変数の名前換えを仮定しない概念と同じように，代入定理の証明の鍵となる概念であるコントロールパスの定義も扱うことができる．一方，pure λ-calculus は代入と β-簡約を扱うことができる．二種類のラムダ計算が定義されるが，pre-λ-calculus で定義された概念を pure λ-calculus へと “リフト” できるかという問題に対して，respectfulness という関数の性質に注目して答えた．また，ラムダ項の簡約についての概念を一般的な形で定義し，それらを用いて β-簡約を定義している．[7] ではこれらを用いて Church-Rosser の定理を HOL 上で証明出来ることを述べている．本論文では，[7] によるラムダ計算の形式化を採用したが，古典的な Church-Rosser の定理ではなく代入定理を検証した点が異なる．

よりよいラムダ計算の形式化のために [15] はノミナル論理を導入し，Barendregt の Substitution lemma や Church-Rosser の定理などいくつかの複雑な定理を検証した．選択公理と互換性のある形式化を工夫することによって HOL 上にノミナル論理に基づく推論の仕組みを組み込むことができるようになった．[15] はノミナル論理の基本概念である atoms や permutation，support を Isabelle/HOL 上でそれぞれ可算無限集合，atom のペアのリスト，atom の集合として表現する．permutation が作用する対象の型は permutation type として一般化される．finitely supported な permutation type として素のラムダ項 `lam` を定義し，permutation の作用を用いて α-同値関係 $\approx$ を定義する．`lam` の α-同値関係による同値類 `lam`$/\approx$ と全単射な集合 $\texttt{lam}_\alpha$ を再帰的に定義することができる．この再帰的な定義から，Barendregt convention[1] (page 26) を仮定したラムダ項に関する帰納法が導かれる．$\texttt{lam}_\alpha$ 上で代入や束縛変数の集合，部分項などを定義するには α-同値による問題があるので，束縛変数について条件を付す再帰的な結合子を定義する．Isabelle/HOL 上のパッケージを作成し，これらの定義を自動的に宣言できるようにしている．[15] のノミナル論理に基づく形式化は本論文で採用した [7] によるラムダ計算の形式化と通じるものがあるが，本論文では古典的な Church-Rosser の定理ではなく代入定理を検証した点が異なる．

[8] はラムダ計算の形式化のためにグラフ簡約を導入している．applicative order reduction な β-簡約とは，引数をまず β-正規形 に変形してから計算す

る．call by value reduction ともいう．normal order reduction は引数の計算は後回しにして計算する．call by need reduction ともいう．normal order reduction に基づいて，引数をコピーするかわりに，共有するようにし，これをグラフで表すのが，グラフ簡約である．グラフ簡約は，α-簡約のためではなく上述の計算効率のために考案されたが，α-簡約の解消にも役立つ．グラフ簡約では，ラムダ式をグラフで表す．束縛変数名は用いず，その代わりに，グラフの辺で結ぶことより，λ-束縛と，束縛される変数が関連付けられる．そのため，α-変換は不要となる．考え方としては，de Bruijn index と同じで，束縛変数名をなくすことができるが，同様の欠点として，束縛変数名がなくなり読みにくくなる．本論文では，グラフによるラムダ式の表現は用いない．本論文では，束縛変数名を用いる形式化を用いた．

[12] は最適な α-簡約の計算量を論じている。このため、閉じたラムダ項をグラフと束縛変数名を与える写像の組で表現している。α-簡約は、束縛変数名を与える写像の変更により表される。α-簡約の解析に特化した形式化であり、β-簡約への適用は示されていない。本論文ではラムダ項を構文的に扱い、この論文ではグラフと束縛変数名の割り当てにより表現している点が異なる。この論文では、β-簡約は扱われていないが、本論文ではそれが主題である。

1.4 本論文の構成

第 2 章では，予備知識として型無しラムダ計算と証明支援システム HOL について説明する．第 3 章では，代入定理の数学的記述とその数学的な証明を説明する．第 4 章では，文脈 α-同値と二種類のラムダ計算について説明する．第 5 章では，代入定理の形式化を与える．第 6 章では，代入定理の形式的な証明を説明する．第 7 章では，この研究の意義と解決した問題を説明する．そして第 8 章でまとめと今後の研究課題について述べる．

第2章　予備知識

2.1　ラムダ計算

ラムダ計算は，ラムダ記法を用いて定義される関数のなす世界について研究するための数学的体系である．型の無いラムダ計算は関数適用を無制限に認めるという一種の理想化された立場でラムダ項 (ラムダ式とも呼ぶ) の考察を行い，Church-Rosser の定理など型の有無に関係なく議論できる問題についてその本質を見るのに適している．この節では型の無いラムダ計算について説明する [13].

$x, y, z, w, t, u, v, \ldots$ を変数とする．変数は可算無限個あるとする．

ラムダ項を次のように再帰的に定義する．

1. 変数はラムダ項である．
2. M がラムダ項，x が変数のとき $(\lambda x.M)$ はラムダ項である．
3. M と N がラムダ項のとき (MN) はラムダ項である．

$M, N, L, P, X, Y, \ldots$ でラムダ項を表す．Λ をラムダ項全体とする．ラムダ項 M を構成していく過程で作られるラムダ項を M の部分項と呼ぶ．ラムダ項について，次の省略記法を用いる．

1. $\lambda x_1 x_2 \cdots x_n.M \equiv (\lambda x_1.(\lambda x_2.(\cdots(\lambda x_n.M)\cdots)))\quad (n \geq 1)$
2. $M_1 M_2 M_3 \cdots M_n \equiv ((\cdots((M_1 M_2)M_3)\cdots)M_n)\quad (n \geq 2)$

ラムダ項 M の中に $(\lambda x.\cdots)$ の形の部分項があるとき，x はこの部分項の中で束縛されているという．M の中に束縛された形で現れている変数を M の束縛変数，束縛されない形で現れる変数を M の自由変数とよぶ．M の自由変数の集合を $\mathrm{FV}(M)$ で表す．

$(\lambda x.\cdots)$ の中の束縛変数を完全に新しい (この式の中に現れない) 変数に置き換えた式を，元の式と同一視する．したがって例えば $\lambda xy.x = \lambda uy.u = \lambda uv.u$ である．これは α-同値関係と呼ばれる．必要ならラムダ式 M_1,M_2,...,M_n の

中でそれらの束縛変数と自由変数が重ならないようにできる．あるラムダ項を α-同値な別の束縛変数を用いるラムダ項と構文上同じものとみなすことを，束縛変数の名前換えと呼ぶ．

代入を次のように定義する．ラムダ項 M と N はそれらの間で束縛変数と自由変数が重ならないとする．M 中のすべての (自由な)x を N で置き換えた結果を $M[x := N]$ とおく．同様に M 中のすべての (自由な)x_1 を N_1,...,x_n を N_n で同時に置き換えた結果を $M[x_1 := N_1, \ldots, x_n := N_n]$ とおく．

1-step β-簡約 $\rightarrow_\beta$ を次のように再帰的に定義する．

1. $(\lambda x.M)N \rightarrow_\beta M[x := N]$.
2. $M \rightarrow_\beta N$ ならば，$(\lambda x.M) \rightarrow_\beta (\lambda x.N)$，$MP \rightarrow_\beta NP$，および $PM \rightarrow_\beta PN$.

$(\lambda x.M)N$ の形のラムダ項を β-redex とよぶ．

1-step β-簡約を有限回繰り返し実行して P から Q が得られるとき，すなわち

$$P = P_1 \rightarrow_\beta P_2 \rightarrow_\beta \cdots \rightarrow_\beta P_n = Q \text{ (ただし } n \geq 1)$$

を満たすラムダ項 P_1,P_2,$\cdots$,P_n があるとき，$M \rightarrow_\beta^* N$ と表し，この関係を β-簡約という．β-簡約は 1-step β-簡約の反射的推移的閉包である．また，

$$P = P_1 \leftrightarrow_\beta P_2 \leftrightarrow_\beta \cdots \leftrightarrow_\beta P_n = Q$$

ただし $M \leftrightarrow_\beta N$ は $M \rightarrow_\beta N$ または $N \rightarrow_\beta M$ と定義する．

を満たす $n \geq 1$ とラムダ項 P_1,P_2,$\cdots$,P_n があるとき，$P =_\beta Q$ と表し，この関係を β-同値関係とよぶ．これは β-簡約 $\rightarrow_\beta^*$ を含む最小の同値関係である．

古典的で重要な次の定理がある．

定理 2.1.1 (Church-Rosser の定理) $M \rightarrow_\beta^* M_i$ $(i = 1, 2)$ ならば，ある N について $M_i \rightarrow_\beta^* N$ $(i = 1, 2)$ が成り立つ．

ラムダ項 M が (β-) 正規形とは $M \rightarrow_\beta N$ となるラムダ項 N が存在しないことと定める．これは M が部分項として β-redex を持たないことと同値である．NF を正規形であるラムダ項全体とする．ラムダ項 N がラムダ項 M の正規形とは N が正規形であって $M \rightarrow_\beta^* N$ となることと定める．ラムダ項 M が弱正規化可能とはあるラムダ項 N が M の正規形であることと定める．WN を弱正規化可能なラムダ項全体とする．項 M が強正規化可能とは無限列 $M_0, M_1, M_2, \ldots$ で $M = M_0$ かつ任意の $i \geq 0$ に対し $M_i \rightarrow_\beta M_{i+1}$ となる列が無いことと定める．

定理 2.1.1 の系として，次のことが言える．

系 2.1.2 (正規形の唯一性) ラムダ項 N_1,N_2 がラムダ項 M の正規形ならば，$N_1 = N_2$.

証明. 定義より，N_1,N_2 は正規形であって $M \to_\beta^* N_1$ かつ $M \to_\beta^* N_2$．定理 2.1.1 より，ある N について $N_i \to_\beta^* N$ $(i = 1, 2)$．N_i は正規形だから，定義より，$N_i \to_\beta N_i'$ となる N_i' は存在しない．$\to_\beta^*$ の定義から，$N_i = N$ $(i = 1, 2)$. よって $N_1 = N_2$．□

系 2.1.3 (正規形への合流性) ラムダ項 N がラムダ項 M の正規形ならば，$M \to_\beta^* M'$ となる任意の M' に対して，$M' \to_\beta^* N$ が成り立つ．すなわち，M' は弱正規化可能である．

証明. 定義より，N は正規形であって $M \to_\beta^* N$．定理 2.1.1 より，ある N' について $N \to_\beta^* N'$ かつ $M' \to_\beta^* N'$ が成り立つ．先と同じ議論により，$N = N'$．よって，$M' \to_\beta^* N$．N は正規形だから，定義より，M' は弱正規化可能．□

系 2.1.4 (β-同値からの合流性) $M_1 =_\beta M_2$ ならば，ある N について $M_i \to_\beta^* N$ $(i = 1, 2)$ が成り立つ．

証明. 仮定から，$M_1 = P_1 \leftrightarrow_\beta P_2 \leftrightarrow_\beta \cdots \leftrightarrow_\beta P_n = M_2$ を満たす $n \geq 1$ とラムダ項 P_1,P_2,$\cdots$,P_n がある．この n に関する帰納法で証明する．$n = 1$ のときは明らか．$n > 1$ のとき，帰納法の仮定より，$P_1 \to_\beta^* N$ かつ $P_{n-1} \to_\beta^* N$ を満たす N がある．もし $P_n \to_\beta P_{n-1}$ なら，$P_n \to_\beta^* N$ より，N は主張を満たす．一方，$P_{n-1} \to_\beta^* P_n$ ならば，定理 2.1.1 より，$N \to_\beta^* N'$ かつ $P_n \to_\beta^* N'$ を満たす N' がある．このとき，N' は主張を満たす．□

系 2.1.5 (β-同値な正規形への β-簡約) N が正規形のとき，$M =_\beta N \Leftrightarrow M \to_\beta^* N$.

証明. $(\Leftarrow)$ は定義から明らか．$(\Rightarrow)$ 系 2.1.4 より，ある N' について $M \to_\beta^* N'$ かつ $N \to_\beta^* N'$ が成り立つ．N は正規形だから，$N = N'$．よって $M \to_\beta^* N$ が成り立つ．□

系 2.1.5 より，ラムダ項 M が弱正規化可能であることと，ある正規形なラムダ項 N があって $M =_\beta N$ であることは同値である．

2.2　HOL

HOL は高階論理の自動証明システムである [19]．演繹的な定理の導出はもとより，定理をその証明に必要な命題に帰着させる仕組みを備えている．これらの演繹や帰着を，自動あるいは対話的に行うことができる．また，型の検証や推論によって誤りを防ぎつつ理論の記述や利用をすることができる．定義や証明の厳密な記述と合わせ，定理の検証に有意義な手段を提供する．Coq など他の定理検証系と同じく，代入定理の検証に十分な能力を備えている．HOL は様々な理論の形式化や検証に用いられている．ラムダ計算においても，[7] 等の先行研究があって，ラムダ計算の新しい定理である代入定理の検証にとって良好な環境にあるシステムと言える．

HOL による証明では，自動証明出来ない一定以上複雑な場合には，証明手法や用いるべき定義・補題などを指示して処理を行わせなければならない．この場合にも細部まで指示しなくても証明を進めることが出来る．しかし使用者が指示すべきものを判断し，適切な証明の方針を考えることが必要である．

HOL で命題を証明する方法には，公理や定理から定理を演繹的に導出する関数を用いる方法と，命題を HOL の Proof manager に goal として登録する方法がある．goal に対し tactic と呼ばれる関数を与えることで，証明すべき命題を別の命題群に帰着させ新しい goal とすることなどを行う．そして，最初に与えた命題が証明可能なことを検証したなら，その命題を定理として扱うことが出来る．

HOL4 は HOL88,HOL90,HOL98 などを継承する最新の HOL システムである．HOL4 の本論文執筆時点での最新バージョンは HOL 4 Kananaskis 7 である．

HOL は関数型プログラミング言語 ML をメタ言語とした処理系である．HOL4 自体，Standard ML(SML) で実装され，実行には SML 実行環境が必要である．HOL の項はメタ環境上では `term` という ML の型を持つ．通常 HOL の項はダブルバッククォートで挟まれる．例えば，‘‘`x /\ y ==> z`‘‘という表現は $x \wedge y \Rightarrow z$ を意味する HOL の項として ML で評価される．

HOL の論理体系には次の特徴がある．

1. 変数の値として関数や述語をとることができる (高階論理).
2. 項が型を持つ.

3. 論理式という別個の文法的カテゴリはなく，bool 型の項がその役目を果たす.

HOL の項には次の表の記号が使われる.

項の種類	HOL での記法	一般的な記法	説明
真	`T`	T	真である
偽	`F`	F	偽である
否定	`~`t	$\neg t$	t でない
論理和	t_1`\/`t_2	$t_1 \vee t_2$	t_1 または t_2
論理積	t_1`/\`t_2	$t_1 \wedge t_2$	t_1 かつ t_2
論理包含	t_1`==>`t_2	$t_1 \Rightarrow t_2$	t_1 ならば t_2
等号	t_1`=`t_2	$t_1 = t_2$	t_1 は t_2 と等しい
全称量化	`!`$x.t$	$\forall x.t$	すべての x について t
存在量化	`?`$x.t$	$\exists x.t$	ある x が存在して t
選択	`@`$x.t$	$\epsilon x.t$	t となるようなある x
条件分岐	`if` t `then` t_1 `else` t_2	$(t \rightarrow t_1, t_2)$	もし t が真 ならば t_1 でなければ t_2

表 1 HOL の論理体系の項

HOL の項の型はメタ環境上では `hol_type` という ML の型を持つ. ‘‘: ...‘‘という表現が HOL の型を表す. 例えば，‘‘`(1,T)`‘‘は `term` という ML の型を持つ ML の項で，‘‘`:num # bool`‘‘という HOL の型を持つ HOL の項として ML で評価される. この型は自然数と真偽値の組の型である. このように，HOL の型システムは ML の型システムと非常に似通っている.

HOL は型を推論できる. 例えば，否定‘‘`~`‘‘は HOL の型‘‘`:bool -> bool`‘‘を持つので，HOL は項‘‘`~x`‘‘に現れる変数‘‘`x`‘‘が HOL の型‘‘`:bool`‘‘を持つと推論する. 推論の結果，型が不整合な場合はエラーとなる. 例えば‘‘`~1`‘‘は‘‘`1`‘‘ の型が‘‘`:num`‘‘なのでエラーとなる.

記号’\’ はラムダ記法で関数項を作る. 例えば，‘‘`\x. x + 1`‘‘は $n \mapsto n+1$ という関数を表す HOL の項で，‘‘`:num -> num`‘‘という HOL の型を持つ.

HOL で形式化される理論の定理は，ML の型 `thm` を持つ ML の値として表現される. 公理やすでにある定理に推論規則を表す関数を適用すると，型 `thm` を持つ新たな定理が得られる. HOL の基本的な論理体系には 5 つの公理と 8 つの原始推論規則がある. 公理は次のように名前を与えられている. `$@` は以下の公理で性質が定義される $('a \rightarrow bool) \rightarrow 'a$ という HOL の型を持つ

関数である.

- BOOL_CASES_AX　$\vdash \forall t.(t = T) \vee (t = F)$
- ETA_AX　　$\vdash \forall t.(\backslash x.tx) = t$
- SELECT_AX　$\vdash \forall P : 'a \to bool\ x : 'a.Px ==> P(\$@\ P)$
- INFINITY_AX　$\vdash \exists f : ind \to ind.\ ONE\_ONE\ f\ \wedge\ \tilde{}(ONTO\ f)$
- IMP_ANTISYM_AX　$\vdash \forall t_1 t_2.(t_1 \Rightarrow t_2) \Rightarrow (t_2 \Rightarrow t_1) \Rightarrow (t_1 \Leftrightarrow t_2)$

推論規則は次のように名前を与えられている.

- ASSUME

$$\frac{}{t \vdash t}$$

- REFL

$$\frac{}{\vdash t = t}$$

- BETA_CONV

$$\frac{}{\vdash (\backslash x.t_1)t_2 = t_1[t_2 = x]}$$

- SUBST

$$\frac{\Gamma_1 \vdash t_1 = u_1, \cdots, \Gamma_n \vdash t_n = u_n, \Gamma \vdash t[t_1, \cdots, t_n]}{\Gamma \cup \Gamma_1 \cup \cdots \cup \Gamma_n| - t[u_1, \cdots, u_n]}$$

- ABS

$$\frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash (\backslash x.t_1) = (\backslash x.t_2)}$$

- INST_TYPE

$$\frac{\Gamma \vdash t}{\Gamma[ty_1, \cdots, ty_n/vty_1, \cdots, vty_n] \vdash t[ty_1, \cdots, ty_n/vty_1, \cdots, vty_n]}$$

- DISCH

$$\frac{\Gamma \vdash t_2}{\Gamma - \{t_1\} \vdash t_1 \Rightarrow t_2}$$

- MP

$$\frac{\Gamma_1 \vdash t_1 \Rightarrow t_2 \quad \Gamma_2 \vdash t_1}{\Gamma_1 \cup \Gamma_2 \vdash t_2}$$

また，推論規則や公理，定理を組み合わせて，定理を導出する推論を表す関数を定義できる．SPEC は定義済みのこのような関数の例である．

- SPEC

$$\frac{\Gamma \vdash !x.t}{\Gamma \vdash t[u/x]}$$

一方，'goal' に tactic と呼ばれる関数を適用することでも証明を進められる．tactic は次の二つの役割がある．

1. goal を subgoal に分解する．

2. subgoal が証明されたときに元の goal が証明される推論を準備する．

goal は `term list * term` の ML の型を持つ ML の項で，証明すべき命題を表し，仮定の論理式リストと結論の論理式の組として評価される．tactic は `goal -> (goal list * (thm list -> thm))` の ML の型を持つ ML の関数で，tactic T を goal g に適用して Tg を評価すると，goal のリストと，thm のリストをとり thm を返す関数の組になる．この goal のリスト内の要素を goal g の subgoal,thm のリストをとり thm を返す関数を justification function と呼ぶ．justification function は，定理を導出する推論を表す関数で，subgoal の表す命題が証明されて定理となったら，それらの定理を取り goal g の表す定理を証明して返す．subgoal のリストが空のときは，Tg を評価して得られる justification function p は，定理の空リストを受け取って g の表す定理を証明して返す．

HOL の goal stack に，ML の関数 `g` で証明したい命題を goal として登録し，ML の関数 `e` で tactic を適用していくと，goal は subgoal に分解され，subgoal が証明されれば分解前の goal が証明される．

tactic には例えば次のものがある．

- STRIP_TAC　　goal の結論部分の ¬ や ⇒，∧，∀ を，subgoal への分割や論理式の仮定への移動によって取り除き，結論をシンプルにする．
- REWRITE_TAC　　受け取った定理のリストを用いて，goal の結論部分を書き換える．
- ASM_REWRITE_TAC　　受け取った定理のリストと goal の仮定部分を用いて，goal の結論部分を書き換える．
- PROVE_TAC　　受け取った定理のリストと goal の仮定部分を用いて，goal の証明を試みる．
- ASSUME_TAC　　受け取った定理を，goal の仮定部分に追加する．

tactical と呼ばれる ML の関数は，tactic を受け取って tactic を返す．tactical には例えば次のものがある．

- THEN　　(*tactic*1 THEN *tactic*2) は，まず *tactic*1 を goal に適用し，その結果生じたすべての subgoal に *tactic*2 を適用する tactic となる．
- REPEAT　　(REPEAT *tactic*) は，*tactic* を失敗するまで繰り返し goal に適用する．

- by (‘*prop*‘ by *tactic*) は，*tactic* で命題 ‘*prop*‘を証明して，goal の仮定部分への ‘*prop*‘の追加を試みる tactic となる．

第3章　代入定理

3.1　定理の数学的証明

この章では，[14] から採った代入定理の記述とその証明全体を与える．ここでは数学的な内容のみを説明し，それらの形式化は第 5 章で与える．

代入定理は任意の弱正規化可能なラムダ項 X に対し MXX が弱正規化可能ならば，任意の弱正規化可能なラムダ項 X, Y に対し MXY もまた弱正規化可能ということを含意する興味深い定理である．

我々は Barendregt convention[1] (page 26) を仮定する．すなわち，全ての束縛変数の名前は互いに異なり，それらは自由変数の名前とも異なっていると仮定する．

定義 3.1.1 任意の自然数 n と $X_1, \ldots, X_n \in WN$ に対し，$MX_1 \ldots X_n \in WN$ となるとき，ラムダ項 M を永続弱正規化可能という．永続弱正規化可能なラムダ項の集合を PWN と表す.

次がこの章の主題となる定理である．

定理 3.1.2 (弱正規化可能性についての代入定理) 任意の $X \in WN$ と任意の i, j $(1 \leq i, j \leq n)$ に対し $M[x_i := X, x_j := X] \in WN$ ならば, 任意の $X_1, \ldots, X_n \in WN$ に対し $M[x_1 := X_1, \ldots, x_n := X_n] \in WN$.

n として 2 をとると，$\forall X \in \mathrm{WN}(MXX \in \mathrm{WN})$ ならば $\forall XY \in \mathrm{WN}(MXY \in \mathrm{WN})$ が示せる．

WN と PWN の間に，WN に属するラムダ項を PWN に属するラムダ項に適用したラムダ項は，WN に属するという有用な関係がある．

補題 3.1.3 $M \in WN$ かつ $N \in PWN$ ならば $MN \in WN$.

この補題は M に関する帰納法によって証明される．

我々は，コントロールパス と 隣接コントロールパスの概念を導入する．

我々は，コントロールパスと隣接変数出現を定義するときは，我々は β-正規形上の α-変換を許さない．これによって束縛変数の出現をその名前で示すことができる．我々は，証明中これらの convention を使用するが，我々の結果は通常の α-変換のあるラムダ計算に対しても成立する．

定義 3.1.4 (コントロールパス) M は NF に含まれ, x と y は M 中の変数出現で，束縛されていても良いとする．M 中 $x \rightsquigarrow_1 y$ とは，M が $x\overrightarrow{N}(\lambda\overrightarrow{y}.L)$ という部分項を持ち，変数出現 x はこの部分項に明示的に示されているもので，変数出現 y は L 中に現れ，変数 y は $\overrightarrow{y}$ に含まれる，ということが成り立っている関係として定義される．

M 中の変数出現の列 $(x_1, \ldots, x_n)(n \geq 1)$ は，任意の $1 \leq i < n$ に対し，M 中 $x_i \rightsquigarrow_1 x_{i+1}$ かつ x_1 は M 中の自由な変数出現であるとき，M 中のコントロールパスであると定義する．

M の自由変数の集合 $\mathcal{S}$ に対し, $(x_1, \ldots, x_n)$ が $x_1 \in \mathcal{S}$ を満たす M 中のコントロールパスであるとき，M 中の $\mathcal{S}$ からのコントロールパスと呼ぶ．

M 中 $x \rightsquigarrow y$ とは，ある M 中のコントロールパス $(x, x_1, \ldots, x_n, y)$ が存在するということが成り立っている関係として定義される．

なお，M 中 $x \rightsquigarrow y$ という関係は，x が M 中自由なとき，M 中 $z \rightsquigarrow_1 w$ という関係の反射的推移的閉包となる．簡単にするため，$(x, x_1, \ldots, x_n, y)$ を表すのにしばしば $x \rightsquigarrow y$ を用いる．

例 3.1.5 $N = \lambda w.x(\lambda v.vvy)(\lambda t.t(\lambda uz.xz))$ とする．$x \rightsquigarrow x$ と $x \rightsquigarrow v$ が成り立つ．さらに，$x \rightsquigarrow_1 t$ と $t \rightsquigarrow_1 z$ が成り立ち，(x, t, z) がコントロールパスとなるので，$x \rightsquigarrow z$ が成り立つ．

定義 3.1.6 (隣接コントロールパス) M は NF に含まれるラムダ項とし，x と y は M 中の変数出現で，束縛されていてもよいし同じ変数でもよいとする．

M が部分項 $x\overrightarrow{N}(\lambda\overrightarrow{z}.y\overrightarrow{L})$ を持ち，変数出現 x と y がこの部分項に明示的に示されているものであるとき，変数出現 x と y は M 中隣接していると呼ぶ．M 中の二つのコントロールパス $(x_1, \ldots, x_n)$，$(y_1, \ldots, y_m)$ は変数出現 x_n と y_m が M 中隣接しているとき，M 中の隣接コントロールパスと呼ぶ．

例 3.1.7 $N = \lambda w.x(\lambda v.vvy)(\lambda t.t(\lambda uz.xz))$ とする．v と y は隣接しているので, $x \rightsquigarrow v$ と $y \rightsquigarrow y$ は隣接コントロールパスである．さらに，N 中の一つ目の v の出現と二つ目の v の出現は隣接しているので，$x \rightsquigarrow v$ と $x \rightsquigarrow v$ は隣接

コントロールパスであるその他の N 中の隣接コントロールパスは: $x \rightsquigarrow x$ と $x \rightsquigarrow v$; $x \rightsquigarrow x$ と $x \rightsquigarrow t$; $x \rightsquigarrow t$ と $x \rightsquigarrow x$; $x \rightsquigarrow x$ と $x \rightsquigarrow z$ である.

[3] の Lemma A13 は次の補題を与えた.

補題 3.1.8 もし $N \in NF$ 中の x からの隣接コントロールパスが存在するなら, $N[x := X] \notin WN$ となる $X \in WN$ が存在する.

ある条件で弱正規化可能であることが, 隣接コントロールパスの非存在を意味することを示すことができる.

補題 3.1.9 N が NF に含まれ, 任意の $X \in WN$ に対し $N[x := X, y := X] \in WN$ となるとき, N 中 $\{x, y\}$ からの隣接コントロールパスは存在しない.

証明. 対偶を示すために, N 中 x, y からの隣接コントロールパスが存在すると仮定する. 補題 3.1.8 で N を $N[y := x]$ と置くと, $N[y := x]$ 中 x からの隣接コントロールパスが存在するから, $N[y := x][x := X] \notin \mathrm{WN}$ となる $X \in \mathrm{WN}$ が存在する. すなわち, $N[x := X, y := X] \notin \mathrm{WN}$. □

代入定理を証明するための主補題は, 補題 3.1.9 の逆である.

補題 3.1.10 (主補題) N 中 $\vec{x}$ からの隣接コントロールパスが存在しないとき, 任意の $\vec{X} \in WN$ に対し, $N[\vec{x} := \vec{X}] \in WN$.

証明. N に関する帰納法で証明する. $\vec{x} = x_1, \ldots, x_n$, $N = \lambda\vec{y}.z\vec{L}$ とする. $U[\overrightarrow{x} := \overrightarrow{X}]$ を U' と略記する.

Case 1. $z \notin \vec{x}$. $N' = \lambda\vec{y}.z\vec{L'}$ となる. L_i $(1 \leq i \leq \mathrm{lh}(\vec{L}))$ 中に $\vec{x}$ からの隣接コントロールパスが存在しないので, 帰納法の仮定から $L'_i \in \mathrm{WN}$ を得る. よって N' は WN に含まれる.

Case 2. $z = x_h$ $(1 \leq h \leq n)$. $L_l = \lambda\vec{u}.v\vec{P} \in \vec{L}$ $(1 \leq l \leq lh(\vec{L}))$ とする. ただし $lh(\vec{u}) = m$. v は L_l 中自由であることが分かる. そうでないなら, $x_h \rightsquigarrow x_h$ と $x_h \rightsquigarrow v$ は N 中 $\{x_h\}$ からの隣接コントロールパスとなる. さらに, $v\vec{P}$ 中 $\{u_i, u_j\}$ $(1 \leq i, j \leq m)$ からの隣接コントロールパスは存在しない. そうでないなら, $v\vec{P}$ 中ある隣接している a, b に対し $u_i \rightsquigarrow a$ と $u_j \rightsquigarrow b$ が成り立ち, $x_h \rightsquigarrow u_i$ かつ $x_h \rightsquigarrow u_j$ なので, $x_h \rightsquigarrow a$ と $x_h \rightsquigarrow b$ が N 中の $\{x_h\}$ からの隣接コントロールパスとなる. 最後に $v\vec{P}$ 中 $\{x_i, u_j\}$ $(1 \leq i \leq n, 1 \leq j \leq m)$ からの隣接コントロールパスは存在しない. そうでないなら, $v\vec{P}$ 中ある隣接している a, b に対し $x_i \rightsquigarrow a$ と $u_j \rightsquigarrow b$ が成り立ち, $x_h \rightsquigarrow u_j$ なので, $x_i \rightsquigarrow a$ と $x_h \rightsquigarrow b$

が N 中の $\{x_i, x_h\}$ からの隣接コントロールパスとなる．よって，$v\vec{P}$ 中 $\vec{x}, \vec{u}$ からの隣接コントロールパスは存在しない．したがって，帰納法の仮定により，任意の $X_1, \ldots, X_n, U_1, \ldots, U_m \in \mathrm{WN}$ に対し $v\vec{P}[\vec{x} := \vec{X}, \vec{u} := \vec{U}] \in \mathrm{WN}$ となる．さらに，任意の $U_1, \ldots, U_m, V_1, \ldots, V_k \in \mathrm{WN}$ に対し $L_l' U_1 \ldots U_m V_1 \ldots V_k$ は $v\vec{P}[\vec{x} := \vec{X}, \vec{u} := \vec{U}]\vec{V} \in \mathrm{WN}$ へと簡約されるので，$L_l' \in \mathrm{PWN}$ を得る．補題 3.1.3 から，$X_h\vec{L'} \in \mathrm{WN}$ を得る．したがって，任意の n と任意の $X_1, \ldots, X_n \in \mathrm{WN}$ に対し，$N' = \lambda\vec{y}.X_h\overrightarrow{L'} \in \mathrm{WN}$ となる．□

これで代入定理を示すことができる．

定理 3.1.2 の証明. $\overrightarrow{x} = x_1, \ldots, x_n$ とし，N は M の β-正規形とする．

補題 3.1.9 により，N 中 $\{x_i, x_j\}$ $(1 \leq i, j \leq n)$ からの隣接コントロールパスは存在しない．したがって $\overrightarrow{x}$ からの隣接コントロールパスは存在しない．補題 3.1.10 により，任意の $\overrightarrow{X} \in \mathrm{WN}$ に対し，$N[\vec{x} := \vec{X}] \in \mathrm{WN}$．$M[\vec{x} := \vec{X}]$ は $N[\vec{x} := \vec{X}]$ に簡約されるので, 任意の $\overrightarrow{X} \in \mathrm{WN}$ に対し，$M[\vec{x} := \vec{X}] \in \mathrm{WN}$. □

第4章　文脈α-同値

この章では，文脈 α-同値，pre-λ-calculus，および pure λ-calculus を数学的に説明する．これらは [7] から採った．その形式化は第 5 章で与えられる．

束縛変数の名前換えの無いラムダ計算を扱うために，[7] は二種類のラムダ計算と文脈 α-同値を与えた．二種類のラムダ計算の一つは pre-λ-calculus で，もう一つは pure λ-calculus である．pre-λ-calculus は束縛変数の名前換えの無いラムダ項を扱う．文脈 α-同値は α-同値関係を定義するために pre-λ-calculus 上に定義される．pure λ-calculus は pre-λ-calculus の α-同値関係による同値類である．

pre-λ-calculus は他の束縛変数の名前換えを仮定しない概念と同じように，コントロールパスも扱うことができる．一方，pure λ-calculus は代入と β-簡約を扱うことができる．

4.1　Pre-λ-Calculus

定義 4.1.1 (Pre-λ-Calculus) 定数に c，変数に v を用いる．pre-λ-calculus の項 t を

$t ::= c|v|tt|\lambda v.t$

によって定義する．pre-λ-calculus の項の集合を Λ_1 で表す．

pre-λ-calculus では，束縛変数の名前換えのないラムダ項を扱う．つまり，

$\lambda xy.x \neq \lambda uy.u \neq \lambda uv.u$

となる．また，

$x_1 = x_2$ かつ $M_1 = M_2$ のとき，そのときに限り $\lambda x_1.M_1 = \lambda x_2.M_2$

が成り立つ．

pre-λ-calculus 上で次のような関数が定義される．$(a \quad =_{\mathrm{def}} \quad b)$ は a を b で定義することを表す．

定義 4.1.2 (項の高さ) $\mathsf{HEIGHT}_1(x) \quad =_{\mathrm{def}} \quad 0,$

$\mathsf{HEIGHT}_1(tu) \quad =_{\mathrm{def}} \quad max\{\mathsf{HEIGHT}_1(t), \mathsf{HEIGHT}_1(u)\} + 1,$
$\mathsf{HEIGHT}_1(\lambda x.u) \quad =_{\mathrm{def}} \quad \mathsf{HEIGHT}_1(u) + 1.$

定義 4.1.3 (自由変数) $\mathsf{FV}_1(x) \quad =_{\mathrm{def}} \quad \{x\},$
$\mathsf{FV}_1(tu) \quad =_{\mathrm{def}} \quad \mathsf{FV}_1(t) \cup \mathsf{FV}_1(u),$
$\mathsf{FV}_1(\lambda x.u) \quad =_{\mathrm{def}} \quad \mathsf{FV}_1(u) - \{x\}.$

4.2　文脈 α-同値

文脈 α-同値を用いて，α-同値関係を定義することができる．[],:: はリストの構成子である．[] は空のリストを表す．xs がリストのとき，$x :: xs$ はリストとなる．例えば，$x :: y :: z :: []$ で x,y,z のリストを表す．

定義 4.2.1 (α-同値) 変数 x, y と変数のリスト xs, ys に対し，文脈 α-同値 $x \overset{xs}{\mathrm{var}} \equiv_\alpha^{ys} y$ は次のようにリストの構造に関して再帰的に定義される．

$$w \overset{x::xs}{\mathrm{var}} \equiv_\alpha^{y::ys} z \quad =_{\mathrm{def}}$$
$$(w = x \wedge z = y \wedge ||xs|| = ||ys||) \vee (w \neq x \wedge z \neq y \wedge w \overset{xs}{\mathrm{var}} \equiv_\alpha^{ys} z),$$
$$w \overset{[]}{\mathrm{var}} \equiv_\alpha^{[]} z \quad =_{\mathrm{def}} \quad (w = z).$$

$x \overset{xs}{\mathrm{var}} \equiv_\alpha^{ys} y$ は，x が xs 中にあり，y が ys 中にあり，それらの最左出現位置が同じであるか，または $x = y$ かつ x は xs 中になく y は ys 中にないことを意味する．

例 4.2.2 $w \neq x_1$ かつ $z \neq y_1$ のとき，または $w = x_1$ かつ $z = y_1$ のとき，
$w \overset{x_1,w,x_2}{\mathrm{var}} \equiv_\alpha^{y_1,z,y_2} z$ が成り立つ．
$w = x_1$ かつ $z \neq y_1$ のとき，または $w \neq x_1$ かつ $z = y_1$ のときは，成り立たない．

例 4.2.3 $w \neq x_i$ かつ $w \neq y_i$ $(i = 1, 2, 3)$ のとき，
$w \overset{x_1,x_2,x_3}{\mathrm{var}} \equiv_\alpha^{y_1,y_2,y_3} w$ が成り立つ．

項 t, u と変数のリスト xs, ys に対し，文脈 α-同値 $t \overset{xs}{\mathrm{var}} \equiv_\alpha^{ys} u$ は次の規則によって t, u の構造に関して帰納的に定義される．

$$x \overset{xs}{\equiv_\alpha^{ys}} y \quad =_{\mathrm{def}} \quad x \overset{xs}{\mathrm{var}} \equiv_\alpha^{ys} y,$$
$$t_1 u_1 \overset{xs}{\equiv_\alpha^{ys}} t_2 u_2 \quad =_{\mathrm{def}} \quad t_1 \overset{xs}{\equiv_\alpha^{ys}} t_2 \wedge u_1 \overset{xs}{\equiv_\alpha^{ys}} u_2,$$
$$\lambda x.t_1 \overset{xs}{\equiv_\alpha^{ys}} \lambda y.t_2 \quad =_{\mathrm{def}} \quad t_1 \overset{x::xs}{\equiv_\alpha^{y::ys}} t_2.$$

この定義では，文脈 α-同値かどうかは項の内部のチェック結果による．ラムダ抽象を通過するたび，リストには新たな変数が付け加わる．最終的に，それぞれの項の中の変数同士の比較によって決定される．

例 4.2.4 $w \neq x_1$ かつ $z \neq y_1$ のとき，または $w = x_1$ かつ $z = y_1$ のとき，
$(\lambda w.(\lambda x_1.w))\ ^{x_2} \equiv_\alpha^{y_2} (\lambda z.(\lambda y_1.z))$ が成り立つ．

項 t, u の α-同値 $t \equiv_\alpha u$ は次のように定義される．

$$t \equiv_\alpha u \quad =_{\mathrm{def}} \quad t\ ^{[]} \equiv_\alpha^{[]} u.$$

例 4.2.5 $w \neq x_1$ かつ $z \neq y_1$ のとき，または $w = x_1$ かつ $z = y_1$ のとき，
$(\lambda x_2.(\lambda w.(\lambda x_1.w)))\ ^{[]} \equiv_\alpha^{[]} (\lambda y_2.(\lambda z.(\lambda y_1.z)))$ が成り立つ．ゆえに，
$(\lambda x_2.(\lambda w.(\lambda x_1.w))) \equiv_\alpha (\lambda y_2.(\lambda z.(\lambda y_1.z)))$ が成り立つ．

関係 $t_1\ ^{xs} \equiv_\alpha^{ys} t_2$ は，xs，ys がそれぞれ重複のない変数のリストのとき，リスト xs にある各変数をリスト ys にある対応する変数で置き換えるとき，t_1 は t_2 と α-同値であることを意味する．

4.3 Pure λ-Calculus

定義 4.3.1 (Pure λ-Calculus) pre-λ-calculus の項の集合の α-同値関係による同値類 $\Lambda_1/\equiv_\alpha$ を，pure λ-calculus の項の集合 Λ と定義する．
t の同値類を表すのに，$\lfloor t \rfloor$ を用いる．各同値類に対して固定された代表元を仮定する．同値類 T の代表元ををを表すのに，$\lceil T \rceil$ を用いる．

次の式を満たす．
$\forall a.\lfloor \lceil a \rceil \rfloor = a$,
$\forall r r'.r \equiv_\alpha r' \Leftrightarrow (\lfloor r \rfloor = \lfloor r' \rfloor)$.
これは 2.1 節で定義した，束縛変数の名前換えのある通常のラムダ計算になる．

例 4.3.2 Λ 中の $\lambda x.x$ は $\{\lambda x.x, \lambda y.y, \lambda z.z, \cdots\} \subseteq \Lambda_1$ を表す．

Λ_1 で定義された関数を Λ で再定義するには，次の三つの段階を踏む．最初に，その関数が α-同値な pre-λ-term を引数に取るとき，値が pre-λ-term なら α-同値に，それ以外ならば等しくなることを証明する．つまり，Λ_1 上の関数 f について

$\forall t_1 t_2.(\ t_1 \equiv_\alpha t_2 \Rightarrow f t_1 \equiv_\alpha f t_2\ if (f:\ \Lambda_1 \to \Lambda_1)\ or\ f t_1 = f t_2\ otherwise\ )$
を示す．このような性質を respectfulness と呼ぶ．FV_1 の場合，
$\forall t_1 t_2.t_1 \equiv_\alpha t_2 \Rightarrow (\mathsf{FV}_1 t_1 = \mathsf{FV}_1 t_2)$
を示せばよい．次に，元の関数と $\lfloor\cdot\rfloor$，$\lceil\cdot\rceil$ を用いて Λ 上の新しい関数を定義する．FV_1 を元に FV を定義する場合，

定義 4.3.3 (Λ における自由変数) $\mathsf{FV}\ t \quad =_{\mathrm{def}} \quad \mathsf{FV}_1 \lceil t \rceil$

と定義する．最後に，元の Λ_1 における関数の定義と同様の形の定理を Λ 上で証明して新しい定義とする．

定義 4.3.4 (Λ における自由変数 (new)) $\mathsf{FV}(x) = \{x\}$,
$\mathsf{FV}(tu) = \mathsf{FV}(t) \cup \mathsf{FV}(u)$,
$\mathsf{FV}(\lambda x.u) = \mathsf{FV}(u) - \{x\}$.

α-同値関係はラムダ項の抽象的関数としての意味が束縛変数の名前によらないで定まるということを表す関係である．pre-λ-term の α-同値関係による同値類である pure λ-term は，ラムダ項を関数として解釈する直観に合い，直観的に，わかり易くラムダ計算を説明し，おおまかに解析できる．抽象的関数としてのラムダ項に期待される $\lambda x.x = \lambda y.y$ というような束縛変数の名前換えによる等式や，Church-Rosser の定理などの良い性質が成り立つ．関数型言語の実装も α-同値を同一視して行われ，また，pure λ-term を表すために，de Bruijn index も作られた．しかし pure λ-term には問題となる場面もある．厳密性に欠ける場合があること，厳密に証明を進める場合には当然その定義に用いている pre-λ-term が必要となること，関数型プログラミング言語でも定義の文面が重要である場合には α-変換してはいけないことなどである．ラムダ項を扱う場合，α-同値関係や束縛変数の名前換えは当然のものとするか，素朴に定義して論を進めることも多いが，厳密には段階を追った慎重な定義をする必要がある．一つの方法はラムダ項の代入を先に定義して，α-同値関係の定義の中で用いることである．しかるに代入の定義自体が，厳密には複雑なものとなる．また pre-λ-term の α-同値も代入を用いて定義はできるが，素直に進むと代入がアルファ同値で安定であることを示すことが困難になり，その解析には，α-同値の補助概念が必ず必要になる．

4.4 代入

pre-λ-term では，代入を定義しても，Church-Rosser の定理が成り立たない．前節で示した pre-λ-term 上の関数をリフトする方法で，pure λ-term での代入を定義する．代入は平行的に行われるように定義する．$[x_1 := e_1, \ldots, x_n := e_n]$ $(n \geq 0)$ は変数とラムダ項の組のリストである．まず Λ_1 で関数を定義する．

定義 4.4.1 (Λ_1 における変数への代入)

$y \lhd_1^v [x := e, x_1 := e_1, \ldots, x_n := e_n]$

$=_{\mathrm{def}}$ **if** $y = x$ **then** e **else** $y \lhd_1^v [x_1 := e_1, \ldots, x_n := e_n]$,

$y \lhd_1^v [\,] \quad =_{\mathrm{def}} \quad y$.

定義 4.4.2 (Λ_1 におけるラムダ項への代入)

$x \lhd_1 [x_1 := e_1, \ldots, x_n := e_n] \quad =_{\mathrm{def}} \quad x \lhd_1^v [x_1 := e_1, \ldots, x_n := e_n]$,

$(tu) \lhd_1 [x_1 := e_1, \ldots, x_n := e_n]$

$=_{\mathrm{def}} \quad (t \lhd_1 [x_1 := e_1, \ldots, x_n := e_n])(u \lhd_1 [x_1 := e_1, \ldots, x_n := e_n])$,

$(\lambda x.u) \lhd_1 [x_1 := e_1, \ldots, x_n := e_n]$

$=_{\mathrm{def}}$ **let**

$x' = \mathsf{variant}\ x\ (\mathsf{FVsubst_1}\ [x_1 := e_1, \ldots, x_n := e_n]\ (\mathsf{FV_1}\ u - \{x\}))$

in

$\lambda x'.(u \lhd_1 [x := x', x_1 := e_1, \ldots, x_n := e_n])$.

ここで，$(\mathsf{FVsubst_1}\ [x_1 := e_1, \ldots, x_n := e_n]\ (\mathsf{FV_1}\ u - \{x\}))$ は，u の x 以外の自由変数に対し，それぞれ $[x_1 := e_1, \ldots, x_n := e_n]$ を代入した結果のラムダ項の自由変数の和集合を意味し，$\mathsf{variant}\ x\ r$ は変数 x と変数の集合 r に対し $\mathsf{variant}\ x\ r \notin r$ となる変数を意味する．

前節でやったように，この関数を Λ にリフトする．respectfulness を示すのは少々複雑である．

定義 4.4.3 (Λ における変数への代入)

$y \lhd^v [x := e, x_1 := e_1, \ldots, x_n := e_n]$

$=_{\mathrm{def}} \quad \lfloor y \lhd_1^v \lceil [x := e, x_1 := e_1, \ldots, x_n := e_n] \rceil \rfloor$,

$y \lhd^v [x := e, x_1 := e_1, \ldots, x_n := e_n]$

$=$ **if** $y = x$ **then** e **else** $y \lhd^v [x_1 := e_1, \ldots, x_n := e_n]$,

$y \lhd^v [\,] = y$.

定義 4.4.4 (Λ におけるラムダ項への代入)

$t \lhd [x := e, x_1 := e_1, \ldots, x_n := e_n]$

$\quad =_{\mathrm{def}} \quad \lfloor \lceil t \rceil \lhd_1 \lceil [x := e, x_1 := e_1, \ldots, x_n := e_n] \rceil \rfloor,$

$x \lhd [x_1 := e_1, \ldots, x_n := e_n] = x \lhd^v [x_1 := e_1, \ldots, x_n := e_n],$

$(tu) \lhd [x_1 := e_1, \ldots, x_n := e_n]$

$= (t \lhd [x_1 := e_1, \ldots, x_n := e_n])(u \lhd [x_1 := e_1, \ldots, x_n := e_n]),$

$(\lambda x.u) \lhd [x_1 := e_1, \ldots, x_n := e_n]$

$= \mathbf{let}$

$\qquad\qquad x' = \mathsf{variant}\ x\ (\mathsf{FVsubst}\ [x_1 := e_1, \ldots, x_n := e_n]\ (\mathsf{FV}\ u - \{x\}))$

$\qquad\qquad \mathbf{in}$

$\qquad\qquad \lambda x'.(u \lhd [x := x', x_1 := e_1, \ldots, x_n := e_n]).$

以下，$\lhd$ の記号は省略し，代入を $(u[x_1 := e_1, \ldots, x_n := e_n])$ と表記する．

第5章　定理の形式化

この章は，第 3 章と第 4 章の内容を形式化する．目標の論理式やラムダ計算，コントロールパスなどの形式化を与える．

5.1　目標の論理式

この節では我々が代入定理の記述を形式化したものを与える．

我々は，第 3 章で与えられたすべての数学的内容を忠実に定式化する．これは我々の仕事によって，[14] で与えられた弱正規化についての代入定理の証明が検証されることを意味する．原論文 [14] において補題 3.1.8 は証明無しで引用されているので，我々は現段階ではこの補題の検証を保留する．

形式化したものと比較するため，ここでもう一度代入定理 (定理 3.1.2) の記述を示す.

弱正規化可能性についての代入定理:

任意の $X \in \mathrm{WN}$ と任意の i, j $(1 \leq i, j \leq n)$ に対し $M[x_i := X, x_j := X] \in \mathrm{WN}$ ならば, 任意の $X_1, \ldots, X_n \in \mathrm{WN}$ に対し $M[x_1 := X_1, \ldots, x_n := X_n] \in \mathrm{WN}$.

我々は次のように HOL でこれを定式化する.

```
g ‘      (~(xl = ([]:(var list)))) ==>
         ((LIST_TO_SET xl) SUBSET (FV M)) ==>
         (CF M) ==>
         (!X xi xj.
            CF X /\ WN X /\ MEM xi xl /\ MEM xj xl ==>
            WN (M <[ [(xi,X); (xj,X)])) ==>
         !XL. EVERY WN XL /\ (LENGTH xl = LENGTH XL) ==>
            WN (M <[ (ZIP(xl,XL)))‘;
```

コード (LIST_TO_SET xl) はリスト xl の要素の集合を表す．コード (FV M) はラムダ項 M の自由変数の集合を表す．コード (s SUBSET t) は集合 s は

t の部分集合であることを表す．コード (CF M) はラムダ項 M は定数を含まないことを表す．コード (WN X) はラムダ項 X は弱正規化可能であることを表す．コード (MEM xi xl) は変数 xi はリスト xl の要素であることを表す．コード (M <[[(xi,X); (xj,X)]) は M 中の xi を X に，xj も X に置き換える代入を表す．コード (EVERY WN XL) はラムダ項のリスト XL の要素はすべて弱正規化可能であることを表す．コード (M <[(ZIP(xl,XL))) は M 中のリスト xl の要素を，リスト XL の対応する要素に置き換える代入を表す．

目標となる論理式は次を意味する．

変数のリスト xl は空であり，xl の要素の集合は，ラムダ項 M の自由変数の部分集合で，M は定数を含まないと仮定する．さらに，任意のラムダ項 X と任意の変数 xi と xj に対し，X が定数を含まず弱正規化可能で xi と xj が xl の要素ならば，M 中の xi に X を，xj にも X を代入をすることによって弱正規化可能なラムダ項が得られることを仮定する．そのとき，任意のラムダ項のリスト XL に対し，全ての XL の要素が弱正規化可能で xl と XL の長さが等しいならば，M 中のリスト xl の要素に，リスト XL の対応する要素を代入することによって弱正規化可能なラムダ項が得られる．

5.2　ラムダ計算

この節では，pre-λ-calculus，pure λ-calculus，そして β-簡約の形式化を示す．この形式化は [7] による HOL のパッケージから採ったものである．

pre-λ-calculus のラムダ項は HOL で次のように定義され，型 term1 と構成子 Con1, Var1, App1, Lam1 を持つ．

```
val _ = Hol_datatype
        ‘ term1 = Con1 of 'a
                | Var1 of var
                | App1 of term1 => term1
                | Lam1 of var => term1 ‘ ;
```

コード (ALPHA N M) は pre-λ terms N と M の α-同値 $N \equiv_\alpha M$ を表す．定理 ALPHA_EQUIV は ALPHA が同値関係であることを示す．

```
val ALPHA_EQUIV = |- EQUIV ALPHA : thm
```

Pure λ-calculus は同値類のための関数 define_quotient_types によって次のように形式化される．この関数は型 term1 と同値関係 ALPHA を取り，型 term1 の ALPHA による同値類に同型な新たな型 term を生成する．

```
val [...] =
    define_quotient_types
    {types = [{name = "term", equiv = ALPHA_EQUIV}], ... };
```

このコードは term を Λ のための新しい型として生成する．関数 Con, Var, App, Lam は構成子関数である．pre-λ-term t に対し，$\lfloor t \rfloor$ はコード (term_ABS t) で表される．pure λ-term T に対し，$\lceil T \rceil$ はコード (term_REP T) で表される．

コード (t <[[(x1,t1); ... ; (xn,tn)]) は pure λ-termt 中の自由変数 x1, ...,xn に pure λ-terms t1,...,tn をそれぞれ代入して得られる pure λ-term を表す．この代入は次の性質を満たす関数として定義される．変数の捕捉を避けるために，この関数は代入を実行する前に束縛変数の名前を変更する．

```
|- (!a s. Con a <[ s = Con a) /\ (!x s. Var x <[ s = SUB s x) /\
   (!t u s. App t u <[ s = App (t <[ s) (u <[ s)) /\
   !x u s.
     Lam x u <[ s =
     (let x' = variant x (FV_subst s (FV u DIFF {x})) in
        Lam x' (u <[ (x,Var x')::s))
```

コード (RED1 R t1 t2) は pure λ-terms t1 と t2 に対して 1-step R-簡約関係 t1 $\rightarrow_R$ t2 があることを表す．

このコードは次が成り立つように定義される．

```
|- (!R t1 t2. R t1 t2 ==> RED1 R t1 t2) /\
   (!R t1 u t2. RED1 R t1 t2 ==> RED1 R (App t1 u) (App t2 u)) /\
   (!R t u1 u2. RED1 R u1 u2 ==> RED1 R (App t u1) (App t u2)) /\
   !R x t1 t2. RED1 R t1 t2 ==> RED1 R (Lam x t1) (Lam x t2)
```

コード (RED R t1 t2) は pure λ-terms t1 と t2 に対して R-簡約関係 t1 $\rightarrow_R^*$ t2 があることを表す．関係 $\rightarrow_R^*$ は関係 $\rightarrow_R$ の反射的推移的閉包である．

このコードは次が成り立つように定義される．

```
|- (!R t1 t2. RED1 R t1 t2 ==> RED R t1 t2) /\
   (!R t1. RED R t1 t1) /\
    !R t1 t3. (?t2. RED R t1 t2 /\ RED R t2 t3) ==> RED R t1 t3
```

コード (REQUAL R t1 t2) は pure λ-terms t1 と t2 が R-同値関係であることを表す．この関係は $\rightarrow_R^*$ を含む最小の同値関係として定義される．

このコードは次が成り立つように定義される．

```
|- (!R t1 t2. RED R t1 t2 ==> REQUAL R t1 t2) /\
   (!R t2 t1. REQUAL R t1 t2 ==> REQUAL R t2 t1) /\
   !R t1 t3. (?t2. REQUAL R t1 t2 /\ REQUAL R t2 t3)
          ==> REQUAL R t1 t3
```

コード (NORMAL_FORM R N) は pure λ-term N が R-正規形，すなわち，N からは 1-step R-簡約が存在しないことを表す．

このコードは次が成り立つように定義される．

```
|- !R a. NORMAL_FORM R a = !a'. ~RED1 R a a'
```

コード (NORMAL_FORM_OF R N M) は pure λ-term N が pure λ-term M の R-正規形，すなわち，M は N と R-同値で，N は R-正規形であることを表す．

このコードは次が成り立つように定義される．

```
|- !R a b. NORMAL_FORM_OF R a b = NORMAL_FORM R a /\ REQUAL R b a
```

コード (BETA_R N M) は pure λ-term M は β-redex N の縮約の結果であることを表す．

このコードは次が成り立つように定義される．

```
|- !x u t. BETA_R (App (Lam x u) t) (u <[ [(x,t)])
```

これらのパッケージで定義済みの関数を用いて，弱正規化可能性を定義する．

コード (WN N) は pure λ-term N は弱正規化可能，すなわち，ある β-正規形に β-同値であることを表す．

このコードは次が成り立つように定義される．

```
|- !M. WN M = ?N. NORMAL_FORM_OF BETA_R N M
```

5.3 コントロールパス

この節ではコントロールパスと隣接コントロールパスの形式化について説明する．

コード (FV1 N) は pre-λ-term N の自由変数の名前の集合としてパッケージで定義されている．Λ_1 で以下のことを定義する．コード (BV1 N) は N の束縛変数の名前の集合を表す．コード (mApp1 X Nl) と (mLam1 yl M) は Nl と yl がそれぞれ $\vec{N}$ と $\vec{y}$ を表すとき，$X\vec{N}$ と $\lambda\vec{y}.M$ を表す．

部分項の概念を形式化するために，indicator を自然数のリストを使って導入する．与えられた pre-λ-term に対し，極大な真部分項のうちの一つを表すのに数値を用いる．もし M が $\lambda x.N$ ならば，0 は N を表す．もし M が NL ならば，0 は N を表し，1 は L を表す．M 中で，indicator$(n_1,\ldots,n_k)$ は，n_1 で表される部分項があり，それに対し n_2 で表されるそのまた部分項があり，$\ldots$，n_k で表されるそのまた部分項，を表す．例えば，$M = t(\lambda uz.xz)$ とする．indicator (1,0,0,1) は M 中で部分項 z を表す．コード (SUBTERM1 M p = N) は indicator p が pre-λ-term M 中で部分項 N を表すことを表す．

コード (FVo1 N) は pre-λ-term N 中の自由な変数出現を表す indicator の集合を表す．コード (CF1 N) は pre-λ-term N は定数を含まないことを表す．これを用いるのは，[14] でのラムダ計算は定数を含まないのに対し，我々が用いる HOL のパッケージでは定数を含んでラムダ計算を定義しているためである．コード (BC_OK N) は pre-λ-term N が Barendregt convention を満たす，すなわち，全ての束縛変数の名前は互いに異なり，それらは自由変数の名前とも異なっていることを表す．

Λ で，コード (mApp X Nl) と (mLam yl M) は Nl と yl がそれぞれ $\vec{N}$ と $\vec{y}$ を表すとき，$X\vec{N}$ と $\lambda\vec{y}.M$ を表す．

以下の定義で，コード M は pre-λ-term である．

関係 $x \rightsquigarrow_1 y$ はコード (control1 M x y) によって次のように定義される．

```
val control1 =
    Define
   `control1 M xo yo =
      (NORMAL_FORM BETA_R (term_ABS M)) /\
      (?M1 Nl L yl x y.
        (  (SUBTERM1 M xo = (Var1 x))
```

```
        /\(SUBTERM1 M yo = (Var1 y))
        /\(SUBTERM1 M M1 =
             (App1 (mApp1 (Var1 x) Nl) (mLam1 yl L)))
        /\(xo=(M1++[0]++(MAP (\x.0) Nl)))
        /\(?yo_in_L.
            ((yo=(M1++[1]++(MAP (\x.0) yl)++yo_in_L)) /\
             (yo_in_L IN (FVo1 L)))
          )
        /\(MEM y yl))
    )‘;
```

第 3 章で M 中 $x \rightsquigarrow y$ という関係を M 中 $(x, x_1, \ldots, x_n, y)$ というコントロールパスが存在することとして定義した．我々はこの概念を変数 x, y に対する概念とする代わりに $(x, x_1, \ldots, x_n, y)$ というパスに対する概念として形式化する．次のコード (control_path M $(x, x_1, \ldots, x_n, y)$) は列 $(x, x_1, \ldots, x_n, y)$ が M 中のコントロールパスであることを表す．

```
val control_path =
    Define
   ‘control_path M xol =
              (NORMAL_FORM BETA_R (term_ABS M)) /\
              (LENGTH xol >= 1) /\
              (!i.
                  (0 <= i /\ i < (LENGTH xol - 1))
                   ==> (control1 M (EL i xol) (EL (SUC i) xol))
              ) /\
              ((EL 0 xol) IN (FVo1 M))‘;
```

次のコード (control_path_from S M $(x_0, \ldots, x_{n-1})$) は列 $(x_0, \ldots, x_{n-1})$ が M 中の S からのコントロールパスであることを表す．

```
val control_path_from =
    Define
   ‘control_path_from s M xol =
        (s SUBSET (FV1 M)) /\
```

```
        (control_path M xol) /\
        ?x.((SUBTERM1 M (EL 0 xol) = (Var1 x)) /\ (x IN s))‘;
```

次のコード (adjacent M x y) は変数出現 x と y が M 中で隣接していることを表す.

```
val adjacent =
    Define
   ‘adjacent M xo yo =
      (NORMAL_FORM BETA_R (term_ABS M)) /\
      (?M1 Nl Ll zl x y.
        (  (SUBTERM1 M xo = (Var1 x)) /\
           (SUBTERM1 M yo = (Var1 y)) /\
           (SUBTERM1 M M1 =
               (App1 (mApp1 (Var1 x) Nl)
                   (mLam1 zl (mApp1 (Var1 y) Ll)))
           )  /\
           (xo=(M1++[0]++(MAP (\x.0) Nl))) /\
           (yo=(M1++[1]++(MAP (\x.0) zl)++(MAP (\x.0) Ll)))
        )
      )‘;
```

次のコード (adjacent_control_paths M $(x_0, \ldots, x_{n-1})$ $(y_0, \ldots, y_{m-1})$) は列 $(x_0, \ldots, x_{n-1})$ と $(y_0, \ldots, y_{m-1})$ が M 中の隣接コントロールパスであることを表す.

```
val adjacent_control_paths =
   Define
   ‘adjacent_control_paths M xol yol =
               (control_path M xol) /\
               (control_path M yol) /\
               (adjacent M (LAST xol) (LAST yol))‘;
```

次のコード
(adjacent_control_paths_from S M $(x_0, \ldots, x_{n-1})$ $(y_0, \ldots, y_{m-1})$) は列

$(x_0, \ldots, x_{n-1})$ と $(y_0, \ldots, y_{m-1})$ が M 中の S からの隣接コントロールパスであることを表す.

第 3 章では明示的には M 中の S からの隣接コントロールパスを定義していない．述語 adjacent_control_paths_from を control_path_from と adjacent_control_paths を組み合わせて次のように定義する．つまり，二つの M 中の S からのコントロールパス $(x_0, \ldots, x_{n-1})$ と $(y_0, \ldots, y_{m-1})$ が M 中の隣接コントロールパスであるとき，$(x_0, \ldots, x_{n-1})$ と $(y_0, \ldots, y_{m-1})$ を M 中の S からの隣接コントロールパスと呼ぶ.

```
val adjacent_control_paths_from =
   Define
   ‘adjacent_control_paths_from s M xol yol =
        (control_path_from s M xol) /\
        (control_path_from s M yol) /\
        (adjacent_control_paths M xol yol)‘;
```

主補題 (補題 3.1.10) を証明するために，これの否定を次のように別に定義する必要がある．この理由については 7.5 節で説明する.

```
val no_adjacent_control_paths_from =
   Define
   ‘no_adjacent_control_paths_from s M =
        ~(?yol zol.
           ( (adjacent_control_paths M yol zol) /\
             (?y.((SUBTERM1 M (EL 0 yol) = (Var1 y)) /\
                  (y IN s) /\
                  (y IN FV1(M))
                 )) /\
             (?z.((SUBTERM1 M (EL 0 zol) = (Var1 z)) /\
                  (z IN s) /\
                  (z IN FV1(M))
                 ))
           )
         )‘;
```

第6章　形式的証明

この章では，我々の行った形式的証明について説明する．

6.1　補題の形式化

HOL のパッケージは証明済みの補題を定理として含んでいる．次は代入定理本体の形式的証明で直接用いる証明済み補題である．

```
- list_CASES;
> val it = |- !l. (l = []) \/ ?t h. l = h::t : thm
```

この補題は，任意のリストは空リストまたはリストの先頭に要素を付け加えたものであることを意味する．

```
- Var_def;
> val it = |- !T1. Var T1 = term_ABS (Var1 T1) : thm
```

この補題は，`Var` の規則である．変数名 `T1` を持つ pre-λ-term の同値類が変数名 `T1` を持つ pure λ-term であることを意味する．

```
- subst_SAME_ONE;
> val it = |- !a x. a <[ [(x,Var x)] = a : thm
```

この補題は，変数 x に変数 x を代入しても変化は無いことを意味する．

```
- subst_SAME_TWO;
> val it = |- !a x t u. a <[ [(x,t); (x,u)] = a <[ [(x,t)] : thm
```

この補題は，代入の定義の仕方から，先に書かれた x への t の代入のみが有効になることを意味する．

```
- REQUAL_TRANS;
> val it = |- !R x y z. REQUAL R x y /\ REQUAL R y z
                        ==> REQUAL R x z : thm
```

この補題は，R-同値関係 $=_R$ は推移的であることを意味する．

```
- REQUAL_SUBSTITUTIVE;
> val it = |- !R. SUBSTITUTIVE R ==> SUBSTITUTIVE (REQUAL R) : thm
```

この補題は，関係 R について，$RMN \Rightarrow R(M[x := L])(N[x := L])$ なら，R を含む最小の同値関係 $=_R$ について，$M =_R N \Rightarrow (M[x := L]) =_R (N[x := L])$ となることを意味する．

```
- BETA_R_SUBSTITUTIVE;
> val it = |- SUBSTITUTIVE BETA_R : thm
```

この補題は，N が β-redex M の縮約の結果であるとき，(N[x := L]) は β-redex (M[x := L]) の縮約の結果であることを意味する．

関数や述語について，その性質が補題として必要な場合，証明して名前を付けることができる．次の代入定理本体の形式的証明で直接用いる補題を証明して名前を付ける．

```
- term_REP_term_ABS_Var1;
> val it = |- !x. term_REP (term_ABS (Var1 x)) = Var1 x : thm
```

この補題は pre-λ-term x の同値類の代表元は元の x であることを意味する．

```
- REQUAL_RED1_BETA_R_SIMUL_SUBSTITUTIVE;
> val it = |- !M N s. REQUAL BETA_R M N
                      ==> REQUAL BETA_R (M <[ s) (N <[ s) :
  thm
```

この補題は，$M =_\beta N$ ならば $(M[\vec{x} := \vec{X}]) =_\beta (N[\vec{x} := \vec{X}])$ であることを意味する．

我々が定義した関数や述語についても，同様に性質を補題として証明して名前を付けることができる．

SUBTERM1 の性質について，次のような補題を証明する．

```
g `(SUBTERM1 N []) = N`;


g `!N vo vo'.
    (SUBTERM1 N (vo++vo')) = (SUBTERM1 (SUBTERM1 N vo) vo')`;
```

```
g `!M N.((((SUBTERM1 M i) = N) /\ (~(M = N)))
         ==> ((HEIGHT1 N) < (HEIGHT1 M)))`;

g `!N' Ll.SUBTERM1 (mApp1 N' Ll) (MAP (\x. 0) Ll) = N'`;

g `!yl N'.SUBTERM1 (mLam1 yl N') (MAP (\x. 0) yl) = N'`;
```

これらの補題は，空の indicator () が部分項として自分自身を指し示すこと，部分項は元の項よりも項の大きさが小さいこと，0 が n 個続く indicator (0,...,0) は $N'\overrightarrow{L}(|\overrightarrow{L}| = n)$ 中で部分項 N' を，0 が m 個続く indicator (0,...,0) は $\lambda\vec{y}.N'(|\vec{y}| = n)$ 中で部分項 N' を，それぞれ指し示すことを意味する．

mApp1,mApp,mLam1,mLam1 の性質について，次のような補題を証明する．

```
g `!XL1 XL2 t.(mApp1 t (XL1 ++ XL2)) = (mApp1 (mApp1 t XL1) XL2)`;

g `!XL1 XL2 t.(mApp t (XL1 ++ XL2)) = (mApp (mApp t XL1) XL2)`;

g `!yl1' yl2' t.(mLam1 (yl1' ++ yl2') t) = (mLam1 yl1' (mLam1 yl2' t))`;

g `!yl1' yl2' t.(mLam (yl1' ++ yl2') t) = (mLam yl1' (mLam yl2' t))`;
```

これらの補題は，pre-λ-term および pure λ-term において，$t(\vec{X_1} + \vec{X_2}) = t(\vec{X_1}\vec{X_2})$，$\lambda(\vec{y_1} + \vec{y_2}).t = \lambda\vec{y_1}\vec{y_2}.t$ であることを意味する．

control1 の性質について，次のような補題を証明する．

```
   |- !N x y vo wo.
        BC_OK N ==>
        ~(x IN BV1 N) ==>
        ~(y IN BV1 N) ==>
        control1 N vo wo ==>
        control1 (term_REP (term_ABS N <[ [(y,Var x)]])) vo wo
 : thm
```

この補題は，N 中 $v \rightsquigarrow_1 w$ であるとき，$N[y := x]$ 中 $v \rightsquigarrow_1 w$ であることを意味する．

control_path の性質について，次のような補題を証明する．

```
g ‘!M M’ aol No.(
    (
      (control_path M’ aol) /\
      (SUBTERM1 M No = M’) /\
      (NORMAL_FORM BETA_R (term_ABS M)) /\
      ((EL 0 (MAP (\x. (No ++ x)) aol)) IN FVo1 M)
    ) ==>
    (control_path M (MAP (\x. (No ++ x)) aol) ))‘;
```

この補題は，M' が正規形 M 中で indicator $(x_1, \ldots, x_n)$ の指し示す部分項であり，$(a_1, \ldots, a_m)$ $(a_i = ({a_i}^1, \ldots, {a_i}^{k_i}))$ が M' 中のコントロールパスかつ a_1 は M 中自由な変数出現を表すとき，$(a'_1, \ldots, a'_m)$ $(a'_i = (x_1, \ldots, x_n, {a_i}^1, \ldots, {a_i}^{k_i})$ が M 中のコントロールパスであることを意味する．

control_path_from の性質について，次のような補題を証明する．

```
g ‘(BC_OK N) ==>
   (control_path_from {x; y} N vol) ==>
   (control_path_from {x} (term_REP (term_ABS N <[ [(y,Var x)])) vol)‘;


g ‘!s s’ N.
 (((s’ SUBSET s) /\ (s SUBSET FV1 N) /\ (control_path_from s’ N yol))
 ==> (control_path_from s N yol))‘;
```

一つめの補題は，$(v_1, \ldots, v_n)$ が N 中 $\{x, y\}$ からのコントロールパスであるとき，$N[y := x]$ 中 $\{x\}$ からのコントロールパスでもあることを意味し，二つめの補題は，s' が s の部分集合で，s が $FV_1(N)$ の部分集合であるとき，N 中 s' からのコントロールパスは N 中 s からのコントロールパスでもあることを意味する．

adjacent の性質について，次のような補題を証明する．

```
g ‘(BC_OK N) ==>
   ({x; y} SUBSET (FV1 N)) ==>
   (adjacent N vo wo) ==>
   (adjacent (term_REP (term_ABS N <[ [(y,Var x)])) vo wo)‘;
```

```
g ‘!M N xo yo No.(
   (
     (adjacent N xo yo) /\
     (SUBTERM1 M No = N) /\
     NORMAL_FORM BETA_R (term_ABS M)
   ) ==>
   (adjacent M (No ++ xo) (No ++ yo)))‘;
```

一つめの補題は，$\{x, y\}$ が $FV_1(N)$ の部分集合であるとき，vo と wo が N 中隣接している変数出現 v と w を表すならば，$N[y := x]$ 中でもやはり隣接している変数出現を表すことを意味し，

二つめの補題は，N が正規形 M 中で indicator No $= (n_1, \ldots, n_k)$ の指し示す部分項であり，xo と yo が N 中で隣接している変数出現を表す indicator $(x_1, \ldots, x_n)$，$(x_1, \ldots, x_n)$ であるとき，$(n_1, \ldots, n_k, x_1, \ldots, x_n)$，$(n_1, \ldots, n_k, x_1, \ldots, x_n)$ が M 中で隣接している変数出現を表すことを意味する．

adjacent_control_paths_from の性質について，次のような補題を証明する．

```
g ‘(BC_OK N) ==>
   (adjacent_control_paths_from {x; y} N vol wol) ==>
   (adjacent_control_paths_from
   {x} (term_REP ((term_ABS N) <[ [(y, (Var x))])) vol wol)‘;

g ‘!s s’ N.
 (((s’ SUBSET s) /\
   (s SUBSET FV1 N) /\
   (~?yol zol. adjacent_control_paths_from s N yol zol))
 ==> (~?yol zol. adjacent_control_paths_from s’ N yol zol))‘;

g ‘!M M’ s aol bol No.(
   (
     (s SUBSET (FV1 M)) /\
```

```
    (adjacent_control_paths_from s M' aol bol) /\
    (SUBTERM1 M No = M') /\
    (NORMAL_FORM BETA_R (term_ABS M)) /\
    (No ++ EL 0 aol IN FVo1 M) /\
    (No ++ EL 0 bol IN FVo1 M)
  )  ==>
  (adjacent_control_paths_from
   s M (MAP (\x. (No ++ x)) aol) (MAP (\x. (No ++ x)) bol))`;
```

一つめの補題は，$(v_1, \ldots, v_n)$ と $(w_1, \ldots, w_m)$ が N 中 $\{x, y\}$ からの隣接コントロールパスであるとき，$N[y := x]$ 中 $\{x\}$ からのコントロールパスでもあることを意味し，二つめの補題は，s' が s の部分集合で，s が $FV_1(N)$ の部分集合であるとき，N 中 s からのコントロールパスが存在しないならば N 中 s' からのコントロールパスも存在しないことを意味する．三つめの補題は，M' が正規形 M 中で indicator $\mathtt{No} = (n_1, \ldots, n_p)$ の指し示す部分項であり，$(a_1, \ldots, a_m)$ $(a_i = ({a_i}^1, \ldots, {a_i}^{k_i}))$ と $(b_1, \ldots, b_{m'})$ $(b_j = ({b_j}^1, \ldots, {b_i}^{l_i}))$ が M' 中 $FV_1(N)$ の部分集合 s からの隣接コントロールパスかつ a_1，b_1 は M 中自由な変数出現を表すとき，$(a'_1, \ldots, a'_m)$ $(a'_i = (n_1, \ldots, n_p, {a_i}^1, \ldots, {a_i}^{k_i}))$ と $(b'_1, \ldots, b'_{m'})$ $(b'_j = (n_1, \ldots, n_p, {b_j}^1, \ldots, {b_i}^{l_i}))$ が M 中 s からの隣接コントロールパスであることを意味する．

no_adjacent_control_paths_from の性質について，次のような補題を証明する．

```
g `!s s' N.
 (((s' SUBSET s) /\ (no_adjacent_control_paths_from s N))
 ==> (no_adjacent_control_paths_from s' N ))`;
```

この補題は，s' が s の部分集合で，N 中 s からのコントロールパスが存在しないならば N 中 s' からのコントロールパスも存在しないことを意味する．

次は代入定理本体の形式的証明で直接用いる補題である．

```
- WN_Var;
> val it = |- !y. WN (Var y) : thm
```

この補題は，変数項は弱正規化可能ということを意味する．

```
- lem_BC';
> val it =
    |- !t xl XL.
         ?t1.
           (t = term_ABS t1) /\
           (BV1 t1 INTER LIST_TO_SET xl = {}) /\
            EVERY (\X. BV1 t1 INTER FV X = {}) XL /\
            BC_OK t1 : thm
```

この補題は，任意の pure λ-term t に対して，ある pre-λ-term t1 があり，t1 の同値類が t で，t1 の束縛変数の集合が変数のリスト xl や pure λ-term のリスト XL の要素の自由変数の集合と共通部分を持たず，t1 が Barendregt convention を満たすことを意味する．

```
- lem_CF_NORMAL_FORM;
> val it = |- !M N. NORMAL_FORM_OF BETA_R N M ==> CF M ==> CF N : thm
```

この補題は，N が M の正規形で，M が定数を含まないならば，N も定数を含まないということを意味する．

```
- lem_CF_term_ABS_CF1;
> val it = |- !N1. CF (term_ABS N1) <=> CF1 N1 : thm
```

この補題は，pre-λ-term とその同値類である pure λ-term について，定数を持っているかどうかは両方同じ答えになることを意味する．

```
- WN_SUB_normform;
> val it =
    |- WN (M <[ [(x,X); (y,X)]) /\ REQUAL BETA_R M N /\
       NORMAL_FORM BETA_R N ==>
       WN (N <[ [(x,X); (y,X)]) : thm
```

この補題は，$M[x := X, y := X]$ が弱正規化可能で，$M =_\beta N$ で，N が正規形のとき $N[x := X, y := X]$ が弱正規化可能になることを意味する．

```
- acpf_expand2;
> val it =
```

```
     [‘‘(!x y.( (MEM x xl /\ MEM y xl) ==> ~(?vol wol.
adjacent_control_paths_from {x;y} N1 vol wol)))‘‘]
    |- ~?vol wol.adjacent_control_paths_from
         (LIST_TO_SET xl INTER FV1 N1) N1 vol wol : thm
```

この補題は，任意の xl の要素 x, y に対し，N1 中の集合 x;y からの隣接コントロールパスが存在しないならば，N1 中の，xl の要素の集合と N1 の自由変数の集合の共通部分からの隣接コントロールパスが存在しないことを意味する．

```
- lem_nacp;
> val it =
    |- ~(?vol wol.
           adjacent_control_paths_from
          (LIST_TO_SET xl INTER FV1 N1) N1 vol wol) ==>
       no_adjacent_control_paths_from (LIST_TO_SET xl) N1 : thm
```

この補題は，N1 中の，xl の要素の集合と N1 の自由変数の集合の共通部分からの隣接コントロールパスが存在しないならば，N1 中の，xl の要素の集合からの隣接コントロールパスの存在の否定を意味する．ここで 5.3 で別に定義した隣接コントロールパスの存在の否定 no_adjacent_control_paths_from を用いている．

補題 3.1.8 は次のように形式化される．

```
val lem_2_7 =
        mk_thm ([],
‘‘!N.
    ( CF1 N ==>
     (NORMAL_FORM BETA_R (term_ABS N)) ==>
     (?yol zol.adjacent_control_paths_from {x} N yol zol) ==>
     ?X.
       ((WN X) /\ (CF X) /\ ~(WN ((term_ABS N) <[ [(x,X)]])))‘‘);
```

補題 3.1.9 は次のように形式化される．

```
- lem_2_8;
```

```
> val it =
    |- !N.
         CF1 N ==>
         BC_OK N ==>
         NORMAL_FORM BETA_R (term_ABS N) ==>
         (!X. WN X /\ CF X ==>
          WN (term_ABS N <[ [(x,X); (y,X)])) ==>
         ~?vol wol. adjacent_control_paths_from {x; y} N vol wol :
  thm
```

補題 3.1.10 は次のものだった.

N 中 $\vec{x}$ からの隣接コントロールパスが存在しないとき，任意の $\vec{X} \in \mathrm{WN}$ に対し，$N[\vec{x} := \vec{X}] \in \mathrm{WN}$.

この補題を円滑に証明するために，二つの点について補題を変更した．第一に，補題 7.3.2 を用いるために，$\vec{X}$ の任意の要素 X_i に対し, X_i はその自由変数として N の束縛変数を持っていてはならないという条件と，Barendregt convention を仮定する必要があった．第二に，7.5 で説明する理由によって，`no_adjacent_control_paths_from` を補題の形式化で用いる必要があった．これによって，次の補題とその形式化に辿り着く.

補題 6.1.1 (新しい主補題) $N \in NF$ 中 $\vec{x}$ からの隣接コントロールパスが存在せず，全ての $\vec{x}$ の要素 x は N の束縛変数の集合に含まれないとする．任意の $\vec{X}$ に対し，全ての $\vec{X}$ の要素 X が弱正規化可能で X の自由変数の集合と N の束縛変数の集合は共通部分を持たないとき，$N[\vec{x} := \vec{X}] \in WN$.

これは次のように形式化される.

```
- lem_2_9;
> val it =
    |- !N xl.
         BC_OK N ==>
         CF1 N ==>
         NORMAL_FORM BETA_R (term_ABS N) ==>
         (BV1 N INTER LIST_TO_SET xl = {}) ==>
         no_adjacent_control_paths_from (LIST_TO_SET xl) N ==>
```

```
        !XL.
          EVERY WN XL /\ (LENGTH xl = LENGTH XL) /\
          EVERY (\X. BV1 N INTER FV X = {}) XL ==>
          WN (term_ABS N <[ ZIP (xl,XL)) : thm
```

主補題の変更によって代入定理の証明は次のように修正される.

定理 3.1.2 の証明.

$\overrightarrow{x} = x_1, \ldots, x_n$ とし，N は M の β-正規形でその束縛変数が $\overrightarrow{X} \in \mathrm{WN}$ の自由変数を含まないとする.

補題 3.1.9 により，N 中 $\{x_i, x_j\}$ $(1 \leq i, j \leq n)$ からの隣接コントロールパスは存在しない．したがって"$\overrightarrow{x} \cap FV(N)$ からの隣接コントロールパスが存在する"ことはない．したがって"$\overrightarrow{x}$ からの隣接コントロールパスは存在しない"．補題 6.1.1 により，$\overrightarrow{X} \in \mathrm{WN}$ に対し，$N[\vec{x} := \vec{X}] \in \mathrm{WN}$．$M[\vec{x} := \vec{X}]$ は $N[\vec{x} := \vec{X}]$ に簡約されるので, 任意の $\overrightarrow{X} \in \mathrm{WN}$ に対し，$M[\vec{x} := \vec{X}] \in \mathrm{WN}$.
□

6.2　定理の形式的証明

第 5 章での形式化や前節で挙げた補題を用いて，代入定理を形式的に証明できる.

代入定理を検証の goal に設定する.

```
- g ‘      (~(xl = ([]:(var list)))) ==>
         ((LIST_TO_SET xl) SUBSET (FV M)) ==>
         (CF M) ==>
         (!X xi xj.
            CF X /\ WN X /\ MEM xi xl /\ MEM xj xl ==>
            WN (M <[ [(xi,X); (xj,X)])) ==>
         !XL. EVERY WN XL /\ (LENGTH xl = LENGTH XL) ==>
            WN (M <[ (ZIP(xl,XL)))‘;

<<HOL message: inventing new type variable names: ’a>>
> val it =
    Proof manager status: 1 proof.
```

```
1. Incomplete goalstack:
     Initial goal:

     xl <> [] ==>
     set xl SUBSET FV M ==>
     CF M ==>
     (!X xi xj.
        CF X /\ WN X /\ MEM xi xl /\ MEM xj xl ==>
        WN (M <[ [(xi,X); (xj,X)])) ==>
     !XL.
       EVERY WN XL /\ (LENGTH xl = LENGTH XL) ==>
        WN (M <[ ZIP (xl,XL))


 : proofs
```

tactic を goal に適用していくと，goal が証明され，定理となる．

```
- e(REPEAT STRIP_TAC);
e(‘WN M‘ by REWRITE_TAC[]);
e(‘(?t h. xl = h::t)‘ by PROVE_TAC[list_CASES]);
e(‘MEM h xl‘ by ASM_REWRITE_TAC[MEM]);
e(‘WN (Var h)‘ by PROVE_TAC[SPEC ‘‘h:var‘‘ WN_Var]);
e(‘CF (Var h)‘ by REWRITE_TAC[CF,Var_def,term_REP_term_ABS_Var1,CF1]);
e(‘CF (Var h) /\ WN (Var h) /\ MEM h xl /\ MEM h xl ==>
 WN (M <[ [(h,Var h); (h,Var h)])‘ by PROVE_TAC[]);
e(PROVE_TAC [subst_SAME_TWO,subst_SAME_ONE]);
e(‘?N. NORMAL_FORM_OF BETA_R N M‘ by PROVE_TAC[WN]);
e(‘NORMAL_FORM BETA_R N /\ REQUAL BETA_R M N‘
 by PROVE_TAC[NORMAL_FORM_OF]);
e(‘?N1.
((N = term_ABS (N1:(’a term1)))
 /\ (((BV1 N1) INTER (LIST_TO_SET xl)) = EMPTY)
 /\ (EVERY (\X. ((BV1 N1) INTER (FV X)) = {}) (XL:((’a term) list)))
 /\ BC_OK N1)‘ by REWRITE_TAC[lem_BC’]);
```

```
e(‘CF N‘ by PROVE_TAC[lem_CF_NORMAL_FORM]);
e(‘CF1 N1‘ by PROVE_TAC[lem_CF_term_ABS_CF1]);
e(‘!x y.( (MEM x xl /\ MEM y xl) ==> ~(?vol wol.
   adjacent_control_paths_from {x;y} N1 vol wol))‘ by REWRITE_TAC[]);
e(STRIP_TAC);
e(STRIP_TAC);
e(STRIP_TAC);
e(‘(!X. CF X /\ WN X ==> WN (M <[ [(x,X); (y,X)]))‘ by PROVE_TAC[]);
e(‘(!X. CF X /\ WN X ==> WN (N <[ [(x,X); (y,X)]))‘
 by PROVE_TAC[WN_SUB_normform]);
e(‘(!X. CF X /\ WN X ==> WN (term_ABS N1 <[ [(x,X); (y,X)]))‘
 by PROVE_TAC[]);
e(ASSUME_TAC (SPEC ‘‘N1:(’a term1)‘‘ lem_2_8));
e(PROVE_TAC[]);
e(‘~(?vol wol.adjacent_control_paths_from
 ((LIST_TO_SET xl) INTER (FV1 N1)) N1 vol wol)‘
 by PROVE_TAC [acpf_expand2]);
e(‘(no_adjacent_control_paths_from (LIST_TO_SET xl) N1)‘
 by PROVE_TAC [lem_nacp]);
e(‘NORMAL_FORM BETA_R (term_ABS (N1:(’a term1))) ‘ by PROVE_TAC[]);
e(ASSUME_TAC
  (SPEC ‘‘xl:(var list)‘‘ (SPEC ‘‘(N1:(’a term1))‘‘ lem_2_9)));
e(‘!XL.((EVERY WN (XL:((’a term) list)))
 /\ (LENGTH xl = LENGTH XL)
 /\ (EVERY (\X. BV1 (N1:(’a term1)) INTER FV X = {}) XL)
==> (WN ((term_ABS N1) <[ (ZIP (xl,XL)))))‘ by PROVE_TAC []);
e(‘((EVERY WN (XL:((’a term) list)))
 /\ (LENGTH xl = LENGTH XL)
 /\ (EVERY (\X. BV1 (N1:(’a term1)) INTER FV X = {}) XL)
==> (WN ((term_ABS N1) <[ (ZIP (xl,XL)))))‘ by PROVE_TAC []);
e(‘(WN (N <[ (ZIP (xl,XL))))‘ by PROVE_TAC[]);
e(‘?N’. NORMAL_FORM_OF BETA_R N’ (N <[ (ZIP (xl,XL)))‘
 by PROVE_TAC[WN]);
```

```
e(‘NORMAL_FORM BETA_R N’ /\ REQUAL BETA_R (N <[ (ZIP (xl,XL))) N’‘
 by PROVE_TAC[NORMAL_FORM_OF]);
e(‘REQUAL BETA_R (M <[ (ZIP (xl,XL))) (N <[ (ZIP (xl,XL)))‘
 by REWRITE_TAC[REQUAL_SUBSTITUTIVE,BETA_R_SUBSTITUTIVE]);
e(PROVE_TAC[REQUAL_RED1_BETA_R_SIMUL_SUBSTITUTIVE]);
e(‘REQUAL BETA_R (M <[ (ZIP (xl,XL))) N’‘
 by PROVE_TAC[REQUAL_TRANS]);
e(‘NORMAL_FORM_OF BETA_R N’ (M <[ (ZIP (xl,XL)))‘
 by PROVE_TAC[NORMAL_FORM_OF]);
e(PROVE_TAC[WN]);

 ...

Goal proved.
 [......] |- WN (M <[ ZIP (xl,XL))
> val it =
    Initial goal proved.
    |- xl <> [] ==>
       set xl SUBSET FV M ==>
       CF M ==>
       (!X xi xj.
          CF X /\ WN X /\ MEM xi xl /\ MEM xj xl ==>
          WN (M <[ [(xi,X); (xj,X)])) ==>
       !XL. EVERY WN XL /\ (LENGTH xl = LENGTH XL) ==>
          WN (M <[ ZIP (xl,XL)) :
  proof

- val thm_2_1_1 = top_thm();
> val thm_2_1_1 =
    |- xl <> [] ==>
       set xl SUBSET FV M ==>
       CF M ==>
       (!X xi xj.
```

```
        CF X /\ WN X /\ MEM xi xl /\ MEM xj xl ==>
        WN (M <[ [(xi,X); (xj,X)])) ==>
    !XL. EVERY WN XL /\ (LENGTH xl = LENGTH XL) ==>
        WN (M <[ ZIP (xl,XL)) :
thm
```

6.3　検証の実行結果

326 の補題と 10119 行の HOL のコードを用い，代入定理を検証した．検証の実行には IBM ThinkPad G41 上で HOL4 Kananaskis 3 で 28 分，HOL4 Kananaskis 7 では 1 時間 49 分かかった．(Mobile Intel Pentium4 Processor 552 (3.46 GHz), 1.0GB Memory, 80GB HDD (5400rpm), Microsoft Windows XP Professional SP3)

第7章　意義と問題解決

この章では，我々の検証の意義と，検証中の問題解決についていくつか説明する．

7.1　ラムダ計算の新しく重要な定理の検証

代入定理は型無しラムダ計算の新しい定理である．型無しラムダ計算はよく研究され，新しい定理を得ることは困難であるので，この分野の新しい定理は，通常の直観力では気づくことのできない意外な概念の細やかな取り扱いを必要とする難解な定理となることが多い．代入定理の記述は簡単だが，その証明は複雑で，記述の中に隠れている細やかな取り扱いを必要とする概念と難しい技法を使用する．代入定理は，[14] で与えられる奥深い結果全体の一部であり，重要な定理である．新しく重要な定理を形式化し検証することは我々の検証の主要な意義の一つである．

7.2　束縛変数の名前換えの無いラムダ項

代入定理の証明の鍵となる概念はコントロールパスである．この概念は束縛変数の名前を使用して定義されている．ゆえに，束縛変数の名前換えの無いラムダ項に対してのみ意味のある定義となる．我々の形式化は，束縛変数の名前換えの無いラムダ項を扱う必要がある．

代入は変数の捕捉を避けるために束縛変数の名前換えを必要とする．β-簡約は代入を用いて定義される．ゆえに，代入と β-簡約は束縛変数の名前換えのあるラムダ項に対してのみ適切に定義される概念である．

我々の形式化では，文脈 α-同値と二種類のラムダ項を用いる．二種類のラムダ項の一つは束縛変数の名前換えの無い pre-λ-terms で，もう一つは pure λ-terms である．pure λ-term は pre-λ-term の α-同値関係に関する同値類である．

我々はコントロールパスを pre-λ-terms に対する概念として形式化し，代入と β-簡約を pure λ-terms に対する概念として形式化する．これら二種類のラムダ項に跨る性質を検証することは，複雑で困難である．

7.3　パラメータ付き永続弱正規化可能性

我々の検証中，変数の捕捉のために $N \in \mathrm{PWN}$ を示すのが難しい．この困難を解決するために，変数名の集合 S に対し新しい概念 PWN_S を導入する．S が自由変数に条件を付けてくれるので，変数の捕捉を避けることができる．集合 PWN_S は集合 PWN より大きな集合だが，我々の検証には十分である．我々は，この節で詳細にこの問題の解決について説明する．

永続弱正規化可能な項 PWN は定義 3.1.1 で次のように定義されている．

任意の自然数 n と $X_1, \ldots, X_n \in \mathrm{WN}$ に対し，$MX_1 \ldots X_n \in \mathrm{WN}$ となるとき，ラムダ項 M を永続弱正規化可能という．永続弱正規化可能なラムダ項の集合を PWN と表す.

補題 3.1.3 は，$M \in \mathrm{WN}$ かつ $N \in \mathrm{PWN}$ ならば $MN \in \mathrm{WN}$ という補題である．この補題を $MN \in \mathrm{WN}$ を示すのに使う場合，$N \in \mathrm{PWN}$ を示す必要がある．すなわち，$\forall L_1 \ldots L_n \in \mathrm{WN}(NL_1 \ldots L_n \in \mathrm{WN})$ を示す必要がある．$\forall L_1 \ldots L_n \in \mathrm{WN}(NL_1 \ldots L_n \in \mathrm{WN})$ を示す場合，$NL_1 \ldots L_n$ について β-簡約をする．この β-簡約は変数が捕捉される代入を引き起こす可能性がある．そしてそれゆえにこの状況を処理することは難しい問題である．

この困難を解決するために，変数名の集合 S に対し PWN_S という概念を導入する．この PWN_S という概念は PWN よりも弱い．$N \in \mathrm{PWN}_S$ を示す場合，その自由変数が S に含まれないような項 $L_1 \ldots L_n \in \mathrm{WN}$ に対してのみ，$NL_1 \ldots L_n \in \mathrm{WN}$ を示せば十分である．N の束縛変数の集合になるように S をとることで，$NL_1 \ldots L_n$ についての β-簡約で変数の捕捉を避けることができる．

定義 7.3.1 ($\mathbf{PWN}_S$) S は変数名の集合とする．

ラムダ項 M は，任意の n と $FV(X_1) \cap S = \ldots = FV(X_n) \cap S = \phi$ であるような任意のラムダ項 $X_0, \ldots, X_{n-1} \in WN$ に対し $MX_0 \ldots X_{n-1} \in WN$ であるとき，S に関してパラメータ付き永続弱正規化可能なラムダ項と呼ぶ．S に関してパラメータ付き永続弱正規化可能なラムダ項の集合を PWN_S と表す.

集合 PWN_S は次のように形式化される．コード (PWN_c S M) は $M \in \mathrm{PWN}_S$ を表す.

```
val PWN_c = Define
‘PWN_c s M =
  !XL.
  (
    ((EVERY WN XL) /\ (EVERY (\X.(((FV X) INTER s) = EMPTY)) XL))
      ==>
    (WN (mApp M XL))
  )‘;
```

補題 3.1.3 に替えて，次の補題を用いる．

補題 7.3.2 ($\mathbf{PWN}_S$ のための新補題) S は変数名の有限集合であるとする．もし $FV(M) \cap S = FV(N) \cap S = \phi$，$M \in WN$，そして $N \in PWN_S$ であるならば，$MN \in WN$.

この補題は次のように形式化される．

```
g ‘(WN M) /\ (FINITE s) /\ (PWN_c s N) /\
 (((FV M) INTER s) = EMPTY) /\ (((FV N) INTER s) = EMPTY)
 ==> (WN (App M N)) /\ (((FV (App M N)) INTER s) = EMPTY)‘;
```

この改善によって，補題 6.1.1 が実際に証明できるようになる．この補題の形式的証明中のある subgoal を例に説明する．次は subgoal の一つで，... は仮定の省略を表す.

```
  PWN_c (BV1 (mApp1 (Var1 z) Ll))
    (term_ABS (mLam1 yl’ (mApp1 (Var1 v) Ll’)) <[ ZIP (xl,XL’))
  ------------------------------------
        ...
     : proof
```

PWN_c の定義の展開を含むいくつかの tactic によって，この subgoal は，次の subgoal に変換される．PWN_c に加えられた新しい条件から，仮定 33 が得られる.

```
REQUAL BETA_R
 (mApp
  (mLam yl' (mApp (Var v) (MAP term_ABS Ll') <[ ZIP (xl,XL')))
   XL1
 )
 (mApp (Var v) (MAP term_ABS Ll') <[ ZIP (xl ++ yl',XL' ++ XL1))
-----------------------------------
             ...
 33.  EVERY (\X. FV X INTER BV1 (mApp1 (Var1 z) Ll) = {}) XL''
             ...
   : proof
```

項　(mApp (mLam yl' (mApp (Var v) (MAP term_ABS Ll') <[ZIP (xl,XL'))) XL1) は，yl' が $y_1, \ldots, y_n$ を表し，コード XL1 が $t_1, \ldots, t_n$ を表し，コード (mApp (Var v) (MAP term_ABS Ll') <[ZIP (xl,XL')) が u を表すとき，$(\lambda y_1 \ldots y_n.u)t_1 \ldots t_n$ を表す．仮定 33 は $(\lambda y_1 \ldots y_n.u)t_1 \ldots t_n$ を β-簡約するときに $y_1, \ldots, y_n$ による変数の捕捉がないことを保証する．この改善は，subgoal を証明することを可能にする．

7.4　コントロールパスの変数代入による保存

我々の検証では，コントロールパスが変数代入によって保存されるという subgoal を証明するのは難しい問題である．この性質は数学的には自明で，第 3 章では証明なしに用いている．しかしながら，その形式的証明は難しい．なぜなら，コントロールパスは，pre-λ-terms，すなわち，束縛変数の名前換えの無いラムダ項に定義された概念である一方，代入は pure λ-terms，すなわち，束縛変数の名前換えのあるラムダ項に定義された概念だからである．それゆえこの性質の形式的な記述は両方のスタイルのラムダ項を混在させて用いており，適切な帰納法を見出すのが難しい．この性質の素朴な形式化は次のように与えられる．

```
|- !N x y vo wo.
     BC_OK N ==>
     ~(x IN BV1 N) ==>
```

```
        ~(y IN BV1 N) ==>
        control1 N vo wo ==>
        control1 (term_REP (term_ABS N <[ [(y,Var x)])) vo wo
 : thm
```

この形式化は，次のような意味になる．Barendregt convention を仮定する．もし N 中 $v \rightsquigarrow_1 w$ ならば，$\lceil \lfloor N \rfloor [y := x] \rceil$ 中 $v \rightsquigarrow_1 w$ である．ここで N は pre-λ-term, $\lfloor N \rfloor$ は N の同値類である pure λ-term, $\lceil \lfloor N \rfloor [y := x] \rceil$ は同値類 $\lfloor N \rfloor [y := x]$ の代表元である pre-λ-term である．

これを証明する直接的な方法は，N に関する帰納法である．しかし，N が $\lceil . \rceil$ and $\lfloor . \rfloor$ の入れ子の内部にあるので，帰納法の仮定を用いることが困難である．この困難を解決するために，次の補題を作る．

```
|- !M1 N x y vo wo Nl L yl x' y' yo_in_L.
     BC_OK N ==>
     ~(x IN BV1 N) ==>
     ~(y IN BV1 N) ==>
     NORMAL_FORM BETA_R (term_ABS N) /\
     (SUBTERM1 N vo = Var1 x') /\
     (SUBTERM1 N wo = Var1 y') /\
     (SUBTERM1 N M1 =
       App1 (mApp1 (Var1 x') Nl) (mLam1 yl L)) /\
     (vo = M1 ++ [0] ++ MAP (\x. 0) Nl) /\
     (wo = M1 ++ [1] ++ MAP (\x. 0) yl ++ yo_in_L) /\
     yo_in_L IN FVo1 L /\ MEM y' yl ==>
     ?Nl L yl x' y'.
     (SUBTERM1 (term_REP (term_ABS N <[ [(y,Var x)])) vo =
       Var1 x') /\
     (SUBTERM1 (term_REP (term_ABS N <[ [(y,Var x)])) wo =
       Var1 y') /\
     (SUBTERM1 (term_REP (term_ABS N <[ [(y,Var x)])) M1 =
       App1 (mApp1 (Var1 x') Nl) (mLam1 yl L)) /\
     (vo = M1 ++ [0] ++ MAP (\x. 0) Nl) /\
     (wo = M1 ++ [1] ++ MAP (\x. 0) yl ++ yo_in_L) /\
```

```
yo_in_L IN FVo1 L /\ MEM y' yl : thm
```

control1 N vo wo の定義は，つまるところある条件 $P_{N,vo,wo}$ を満たす M, Nl, L, Ll, x, y が存在するということであった．ここで M は indicator である．一つ目の補題は，N, vo, wo に対し条件 $P_{N,vo,wo}$ を満たす M, Nl, L, Ll, x', y' が存在するとき，$N[y := x], vo, wo$ に対し $M', Nl', L', Ll', x'', y''$ があって条件 $P_{N[y:=x],vo,wo}$ を満たすということである．一方，二つ目の補題は，$M1, Nl, L, Ll, x', y'$ が N, vo, wo に対し条件 $P_{N,vo,wo}$ を満たすならば，$N[y := x], vo, wo$ に対し Nl', L', Ll', x'', y'' があって，$M1, Nl', L', Ll', x'', y''$ が条件 $P_{N[y:=x],vo,wo}$ を満たすということである．以上のように二つ目の補題は，一つ目の補題より強い命題である．これは変数の代入によって部分項の位置は変わらないので，実は $M = M' = M1$ と考えてよい，としたものである．よって二つ目の補題からは一つ目の補題が導かれ，二つ目の補題は一つ目の補題と等価であって，indicator M1 に関する帰納法で証明できる．この技法によって困難が解決する．

7.5　隣接コントロールパスの存在の否定

数学では最初にある概念を定義し，そして別の概念を最初のものを用いて定義する．自由変数についての条件はしばしば細やかな取り扱いが必要であるにもかかわらず暗黙的で見過ごされがちである．最初の概念がある自由変数についての条件を仮定し，二番目の概念が暗黙的に別の自由変数についての条件を仮定するといったことも起こりうる．この困難が adjacent_control_paths_from の定義に発生する．この定義は命題の仮定において肯定的に用いられるときは上手くいくが，その否定を仮定において用いると，暗黙的な自由変数についての条件のため，機能しない．この困難を解決するために，我々はその否定を表現する述語 no_adjacent_control_paths_from を導入する．我々は，この節で詳細にこの問題の解決について説明する．

第 3 章の数学的証明では，“M 中の S からのコントロールパス” は定義されていたが，“M 中の S からの隣接コントロールパス” は定義されていなかった．通常の数学的な記述法に従って，第 3 章では “M 中の S からの隣接コントロールパス” は “M 中の S からのコントロールパス” と隣接コントロールパスの二つの概念の組み合わせで暗黙的に定義されている．定義 3.1.4 の “M

中の S からのコントロールパス" の定義によると，S は自由変数の集合と仮定されている．

我々の形式化は第 3 章に忠実に従っている．最初に control_path_from を次のように定義する．定義 3.1.4 にも含まれている条件なので，(s SUBSET (FV1 M)) を仮定している．

```
val control_path_from =
    Define
   `control_path_from s M xol =
        (s SUBSET (FV1 M)) /\ (control_path M xol) /\
         ?x.((SUBTERM1 M (EL 0 xol) = (Var1 x)) /\ (x IN s))`;
```

述語 adjacent_control_paths_from を control_path_from と adjacent_control_paths を組み合わせて次のように定義する．

```
val adjacent_control_paths_from =
   Define
   `adjacent_control_paths_from s M xol yol =
        (control_path_from s M xol) /\
        (control_path_from s M yol) /\
        (adjacent_control_paths M xol yol)`;
```

しかしながら，述語 adjacent_control_paths_from はその否定を用いると，上手く機能しない．なぜなら control_path_from が自由変数についての条件を仮定していて，それは "M 中の S からのコントロールパス" に対して第 3 章で暗黙的に仮定されている自由変数についての条件とは違うためである．例えば，"M 中の S からのコントロールパス" の否定を仮定に含む命題を証明するとき，余分な自由変数についての条件を示す必要があり，形式的証明が進まなくなる．

この困難を解決するために，adjacent_control_paths_from の否定を別に定義する述語 no_adjacent_control_paths_from を導入する．そのコードは次のようになる．

```
val no_adjacent_control_paths_from =
   Define
   `no_adjacent_control_paths_from s M =
```

```
~(?yol zol.
   ( (adjacent_control_paths M yol zol) /\
     (?y.((SUBTERM1 M (EL 0 yol) = (Var1 y)) /\
          (y IN s) /\
          (y IN FV1(M))
         )) /\
     (?z.((SUBTERM1 M (EL 0 zol) = (Var1 z)) /\
          (z IN s) /\
          (z IN FV1(M))
         ))
   )
 )‘;
```

第 3 章の主補題である補題 3.1.10 がその例である．その記述は “M 中の S からのコントロールパス” の否定を仮定に含む．この改善によって，この補題の形式的証明を完了することができる．

第8章　結論

8.1　まとめ

本論文では，高階論理証明システム HOL を用いて代入定理を形式化し検証した．我々は [14] の数学的内容を忠実に形式化した．束縛変数の名前換えの無いラムダ項を扱うために，[7] のパッケージによって文脈 α-同値を使用した．我々の検証はまた，[7] で与えられた文脈 α-同値のアイデアとそれに基づく HOL のパッケージが束縛変数の名前換えの無いラムダ項の有力な形式化であることを示した．

我々の検証には 5 点の成果があった．第一に，我々は新しく重要な定理を証明した．代入定理はラムダ計算の新しく重要な定理なので，この検証は意義がある．第二に，束縛変数の名前換えのあるラムダ項だけでなく，束縛変数の名前換えの無いラムダ項も必要とする証明を検証した．第三に，変数の捕捉による困難を解決するために，束縛変数の名前の集合 S に対し，PWN_S という概念を導入した．第四に，代入によりコントロールパスが保たれることを検証するために帰納法に適切な変数を選択する工夫を用いた．第五に，隣接コントロールパスの存在の否定を表現するために自由変数条件を正しく表すような新たな述語を導入した．この検証では，10119 行の HOL のコードと 326 の補題を用いた．

8.2　今後の課題

本論文で用いた束縛変数と自由変数を形式的証明で扱う手法は，それらの変数が重要であるような他の定理を検証するのに役立つであろう．特に，我々の手法を用いて，[14] の強正規化可能性についての代入定理や [3] の Lemma A13 の検証ができよう．それらは両方ともラムダ計算の新しく意義深い定理で，我々の形式化で重要な概念であったコントロールパスの概念を用いている．それらの検証は我々の今後の課題である．

謝辞

本研究を進めるにあたって御指導頂いた，龍田真教授に感謝いたします．龍田先生には総研大博士課程在籍時の指導教官としてのみならず，大学学部時代から現在に至るまで親身に指導していただきました．本研究においても私の至らぬ所を助けて頂き，指導に多大な労力を費やしてくださいました．Logic セミナーにおいて御指導頂いた胡 振江教授，金沢誠准教授に感謝いたします．両先生に多くの助言を頂いて，本研究を進めることが出来ました．佐藤健教授と井上克巳教授には，龍田先生，胡先生，金沢先生と共に本論文の審査委員に就いていただきました。お忙しい中審査して下さった先生方に感謝いたします．Logic セミナーなどの研究活動では，他にも多くの先生方，先輩方，学友に多くのことを教わりました．そのことを改めて思い返し，全ての皆様に感謝いたします．また，特任研究員としての国立情報学研究所への在籍が，研究を継続できた大きな要因であることは間違いありません．大学院生として教育を受けた時期のことも含め，国立情報学研究所と研究所の皆様に感謝いたします．最後になりますが，長い研究生活を支え，見守ってくれた両親を始めとする家族に改めて感謝します．

参考文献

[1] Barendregt, H. P.: *The Lambda Calculus: Its Syntax and Semantics*, North-Holland (1984).

[2] Curry, H.B. and Feys, R.: *Combinatory Logic*, North-Holland, (1958).

[3] Dezani-Ciancaglini, M., Honsell, F. and Motohama, Y.: Compositional Characterization of λ-terms using Intersection Types, *Theoretical Computer Science*, Vol. 340, No. 3, pp. 459–495 (2005).

[4] Gonthier, G.: Formal Proof—The Four-Color Theorem, *Notices of the American Mathematical Society*, Vol. 55, No. 11, pp. 1382–1393 (2008).

[5] Gordon, A. D. and Melham, T.: Five axioms of alpha conversion. In J. von Wright, J. Grundy, and J. Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs '96*, volume 1125 of *Lecture Notes in Computer Science*, pages 173-190, Springer-Verlag, (1996).

[6] Hindley, J.R. and Seldin, J. P.: *Introduction to Combinators and lambda Calculus*, Cambridge University Press, (1986).

[7] Homeier, P.V.: A proof of the church-rosser theorem for the lambda calculus in higher order logic, *Supplemental Proceeding of 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2001)*, pp. 207–222 (2001).

[8] Hudak, P.: Conception, evolution, and application of functional programming languages, *ACM Computing Surveys*,21 (3) (1989) 359–411.

[9] Norrish, M.: Mechanising Hankin and Barendregt using the GordonMelham axioms, *the second ACM SIGPLAN workshop MEchanized Reasoning about Languages with varIable and Names*, (MERLIN 2003).

[10] McKinna, J. and Pollack, R.: Pure Type Systems Formalized, *Lecture Notes in Computer Science*, Vol. 664, pp. 289–305 (1993).

[11] Shankar, N.: A mechanical proof of the Church-Rosser theorem, *JACM*, Vol. 35, No. 3, pp. 475–522 (1988).

[12] Statman R.: On the complexity of alpha conversion, *The Journal of Symbolic Logic*, vol. 72, no. 4, (2007), 1197-1203.

[13] 高橋正子: 計算論 計算可能性とラムダ計算, 近代科学社 (1991).

[14] Tatsuta, M. and Dezani-Ciancaglini, M.: Normalisation is Insensible to lambda-term Identity or Difference, *Proceedings of Twenty First Annual IEEE Symposium on Logic in Computer Science*, pp. 327–336 (2006).

[15] Urban, C.: Nominal Techniques in Isabelle/HOL, *Journal of Automated Reasoning*, Vol. 40, No. 4, pp. 327–356 (2008).

[16] Vestergaard, R. and Brotherston, J.: A Formalised First-Order Confluence Proof for the lambda-Calculus using One-Sorted Variable Names, *Information and Computation*, Volume 183, Number 2, pp. 212–244 (2002).

[17] The Agda Wiki, `http://wiki.portal.chalmers.se/agda/pmwiki.php`, 2011-11-10.

[18] The Coq Proof Assistant, `http://coq.inria.fr/`, 2011-11-10.

[19] HOL4 Kananaskis 7 (online), `http://hol.sourceforge.net/`, 2011-11-10.