# 依存型を含むマルチステージプログラミングの型理論

河田 旺　五十嵐 淳

実行中にプログラム自体を生成・実行するマルチステージプログラミングにおいて，生成されたプログラムの安全性を保証することは重要な課題である．本研究では，Hanada,，Igarashi による既存のマルチステージプログラミング言語を拡張し，依存型を扱えるようにすることで，生成されたプログラムの安全性をより強固に保証する型システムを提案する．依存型を導入するには型同士の等価関係を定義することが必要だが，コード化されたプログラムの型の間の等価関係は，値をコードの中に埋め込むための CSP と呼ばれる操作と関連するときに自明ではない．本研究では，型とカインドについて暗黙に CSP を認め，それに対応する等価関係のための規則を追加することで型の間の等価関係を定義した．更に提案する型システムに望ましい性質として型システムの健全性と合流性，強正規化性を挙げ，その証明の見通しについて議論する．

## 1　はじめに

### 1.1　依存型

依存型とは値に依存する型である．依存型を使えば，単純型付きラムダ計算では表現できない，長さ $n$ の整数を要素とする配列の型 Vect $n$ や，範囲を 0 から $n-1$ までに制限した整数の型 Index $n$ を表現することができる．

依存型の応用例として配列外参照の防止 [14] が挙げられる．配列外参照とは配列の範囲外のメモリ領域へのアクセスを指し，コンピュータセキュリティ上の深刻なセキュリティホールになりうる．

依存型をもつ型付ラムダ計算 λLF [1] では，Vect は Vect :: (Int → ∗) と宣言され，Index は Index :: (Int → ∗) のように宣言される．∗ は型を表すカインドで，この宣言はそれぞれ，整数 $n$ を受け取り長さ $n$ の配列の型 Vect $n$ を返す，整数 $n$ を受け取り範囲を 0 から $n-1$ までに制限した整数の型 Index $n$ を返すという意味である．この 2 つの型を使うと，長さ $n$ の Int 型の配列から要素を読みだす関数 read は read : ($\Pi n$ : Int.(Vect $n$ → Index $n$ → Int)) と表現できる．Π は依存関数型のための記号であり，$\Pi x : T_1.T_2$ は $T_1$ 型の引数を受け取り $T_2$ を返す関数の型を表す．ただし，$T_2$ は $x$ を含んでもよく，返り値型が引数 $x$ に依存することを表す．この型を持つ関数 read による配列へのアクセスは決して配列外参照を起こさないことが保証できる．

### 1.2　マルチステージプログラミング

マルチステージプログラミングとはランタイムにプログラムのコード片を組み合わせ，生成，実行できるプログラミングのパラダイムの一つである．これらの操作によりプログラムを部分的に最適化し，高速化することが可能になる．

本研究ではマルチステージプログラミングを実現する方法として，bracket，escape，run，cross-stage persistence(CSP) といった機構 [11] を備える型付きプログラミング言語を考える．bracket，escape，run はそれぞれプログラムの断片をコードと呼ばれる値として得る操作，プログラムを実行しその結果得られるコード値を bracket 中に埋め込む操作，コード値をプログラムとして実行する操作である．cross-stage

Type Theory for Multi-stage Programming with Dependent Types
Akira KAWATA, Atushi IGARASHI, 京都大学大学院情報学研究科, Graduate School of Infomatics, Kyoto University.

persistence は bracket 中にその bracket の外で定義された変数または項を埋め込む操作である．

これらの操作の型安全性を保証する，特に，run が実行するコードが型安全であることを静的に保証するための型システムとして様々なもの [11] [6] [10] が提案されている．Hanada と Igarashi [6] は，ステージ変数と呼ばれる，各コードがどこでコード化されたのかを表す情報と，ステージ変数に関する抽象化を使って bracket，escape，run，cross-stage persistence の全てを含む言語 $\lambda^{\triangleright\%}$ のための型システムを提案し，その型安全性を示している．

### 1.3 依存型を含むマルチステージプログラミング

本研究では $\lambda^{\triangleright\%}$ に λLF を組み込み，依存型を含むマルチステージプログラミングを扱える型システム $\lambda^{\mathrm{MD}}$ を実現する．$\lambda^{\mathrm{MD}}$ ではマルチステージプログラミングで生成，実行するコードに依存型を用いて型付けすることができる．

例えば，ベクトルの長さを受け取って，2 つのベクトルを受け取って 2 つのベクトルの和を返す関数のコードを返す関数 add-vec を考える．add-vec の型は次のようになる．ここで $\forall\alpha.\triangleright_\alpha T$ はコードが実行可能であることを表している．

add-vec :($\Pi x$ : Int.

$\forall\alpha.\triangleright_\alpha((\mathrm{Vect}\ x) \to (\mathrm{Vect}\ x) \to (\mathrm{Vect}\ x)))$

add-vec ではコードを生成する際にベクトルの長さ $x$ を使うことができるので，add-vec が生成したコードの実行時には配列の境界検査が省略できる．さらに，$x$ の情報を使ってベクトルの各要素を足し合わせるループを展開すれば，ループによる分岐命令や jmp 命令が削除できるので大幅な高速化が見込まれる．

本論文の構成を説明する．2 節では $\lambda^{\mathrm{MD}}$ の形式的な定義を行う．2.1 節では構文，2.2 節では $\lambda^{\mathrm{MD}}$ の判断，2.3 節では簡約規則，2.4 節では型システムについてそれぞれ説明する．3 節では $\lambda^{\mathrm{MD}}$ の例とその型付け導出について説明する．4 節では $\lambda^{\mathrm{MD}}$ の満たすべき性質について述べる．5 節では関連研究について述べた後に，6 節で結論と今後の課題を論じる．

## 2 $\lambda^{\mathrm{MD}}$ の形式化

この節では $\lambda^{\mathrm{MD}}$ について詳細に述べる．2.1 節から $\lambda^{\mathrm{MD}}$ の構文, 簡約規則, 型システムを定義する．

$\lambda^{\mathrm{MD}}$ は λLF と $\lambda^{\triangleright\%}$ 双方の拡張になっている．λLF と $\lambda^{\triangleright\%}$ は CSP を除いて機能的に互いに干渉する部分がなく，組み合わせるのは比較的容易であった．ただし，$\lambda^{\triangleright\%}$ の CSP に関する部分は λLF と組み合わせるにあたって非自明な考察が必要であった．

### 2.1 構文

$\lambda^{\mathrm{MD}}$ の構文を定義する．

$$
\begin{aligned}
t &::= x \mid \lambda x:T.t \mid t\ t \mid \blacktriangleright_\alpha\ t \mid \blacktriangleleft_\alpha\ t \mid \Lambda\alpha.t \mid t\ \epsilon \mid \%_\alpha t \\
T &::= X \mid \Pi x:T.T \mid T\ t \mid \triangleright_\alpha T \mid \forall\alpha.T \\
K &::= * \mid \Pi x:T.K \\
\Gamma &::= \emptyset \mid \Gamma, x:T@A\ (x \notin \mathrm{FV}(\Gamma)) \\
&\quad \mid \Gamma, X :: K@A\ (X \notin \mathrm{FV}(\Gamma))
\end{aligned}
$$

ここで, $t$ は項, $x$ は変数，$T$ は型, $X$ は型変数，$K$ はカインド, $\Gamma$ は環境を表す．また $\alpha$, $\beta$ はステージ変数, $A$, $B$ はステージであり 0 個以上のステージ変数の列である．$\epsilon$ は 0 個のステージ変数からなるステージを表す．$\lambda x:T.t$ は $t$ 中の自由な $x$ を束縛し，$\Pi x:T_1.T_2$ は $T_2$ 中の自由な $x$ を束縛する．同様に $\Lambda\alpha.t$ は $t$ 中の自由な $\alpha$ を束縛し，$\forall\alpha.t$ は $t$ 中の自由な $\alpha$ を束縛する．

まず，$\lambda^{\mathrm{MD}}$ の項について直感的な説明を与える．既に述べたように，$\lambda^{\mathrm{MD}}$ は (簡略化のためのいくつかの制限はあるが)$\lambda^{\triangleright\%}$ の拡張であり，$\lambda^{\triangleright\%}$ で依存型を扱えるようにしたものである．$\lambda^{\mathrm{MD}}$ の項はほぼ $\lambda^{\triangleright\%}$ と同じものであり，単純型付きラムダ計算の構文要素に加えて，5 つの追加要素がある．$\blacktriangleright_\alpha\ t$ は $t$ をコード化したものを表す．$\blacktriangleleft_\alpha\ t$ は escape で $\blacktriangleright_\alpha$ の中で使われる．$t$ を評価して得られたステージ $\alpha$ のコードを周りのコード値に埋め込む．$\Lambda\alpha.t$ はステージ変数に関する抽象，$t\ \epsilon$ は $\epsilon$ によるステージ変数の空列によるインスタンス化を表す．$\%_\alpha\ t$ は CSP を表すための記号である．

$\lambda^{\triangleright\%}$ では任意のステージでインスタンス化できるステージ抽象と $\epsilon$ のみでインスタンス化できるステー

ジ抽象の 2 種類のステージ抽象が存在したが，$\lambda^{\mathrm{MD}}$ には $\epsilon$ のみでインスタンス化できるステージ抽象しか存在しない．これは型システムを簡略化するための制限である．

例えば，$\blacktriangleright_\alpha\ ((\lambda x.x)(\lambda y.y))$ はコード化された計算の例であり，$(\Lambda\alpha.\ \blacktriangleright_\alpha\ ((\lambda x.x)(\lambda y.y)))\ \epsilon$ として後で説明するように実行すると $\lambda y.y$ になる．また，$\blacktriangleright_\alpha\ ((\lambda x.\lambda y.x)\ (\blacktriangleleft_\alpha\blacktriangleright_\alpha\ ((\lambda x.x)(\lambda y.y)))\ (\lambda z.z))$ はコード化された計算を escape する例であり，簡約すると $\blacktriangleright_\alpha\ ((\lambda x.\lambda y.x)\ (\lambda y.y)\ (\lambda z.z))$ になる．

次に，$\lambda^{\mathrm{MD}}$ の型について直感的な説明を与える．$\lambda^{\mathrm{MD}}$ の型は $\lambda$LF 由来のものと $\lambda^{\triangleright\%}$ 由来のものの 2 種類にわけられる．$X$ は型変数であり，$\lambda$LF 由来である．$\Pi x : T.T$，$T\ t$ は $\lambda$LF 由来の型である．$\Pi x : T_1.T_2$ は値 $x$ に依存する型を表し，$T_2$ に $x$ が自由に出現しないときは $T_1 \to T_2$ と書かれることがある．$T\ t$ は値に依存する型 $T$ に値 $t$ を適用した結果の型を表し，Vect 10 などが該当する．$\triangleright_\alpha T$，$\forall\alpha.T$ は $\lambda^{\triangleright\%}$ 由来の型である．$\triangleright_\alpha T$ はステージ $\alpha$ における型 $T$ のコードの型を表し，$\forall\alpha.T$ は，ステージ変数に関する抽象化に対して与えられる型で，項がステージ変数 $\alpha$ について閉じていることを表す．

$\lambda^{\triangleright\%}$ ではステージ変数に関する抽象化に対して与えられる型には，任意のステージでインスタンス化できるステージ抽象につくものと $\epsilon$ のみでインスタンス化できるステージ抽象につくものの二種類が存在したが，$\lambda^{\mathrm{MD}}$ には $\epsilon$ のみでインスタンス化できるステージ抽象につく型しか存在しない．これは項と同じ理由である．

最後に，$\lambda^{\mathrm{MD}}$ のカインドについて直感的な説明を与える．$*$ は値につく型につくカインドである．例えば，Int や Vect 10 にはカインド $*$ がつく．$\Pi x : T.K$ は型 $T$ の値を受け取ってカインド $K$ を返す型につくカインドである．例えば Vect にはカインド $\Pi x : \mathrm{Int}.*$ がつく．ただし $K$ に $x$ が自由に出現しないときは $T \to K$ と書かれることがある．

### 2.2 判断

$\lambda^{\mathrm{MD}}$ には 9 種類の判断がある．

$$\Gamma \vdash K\ \ \mathrm{kind}@A$$

はカインド $K$ が環境 $\Gamma$ の下でステージ $A$ において適格 (well-formed) なカインドであることを表す．

$$\Gamma \vdash T :: K@A$$

は型 $T$ が環境 $\Gamma$ の下でステージ $A$ において $K$ のカインドがつくことを表す．

$$\Gamma \vdash t : T@A$$

は項 $t$ が環境 $\Gamma$ の下でステージ $A$ において $T$ の型がつくことを表す．

$$t_1 \longrightarrow_\beta t_2$$

は項 $t_1$ が一回の $\beta$ 簡約で $t_2$ になることを表す．

$$t_1 \longrightarrow_{\blacktriangleleft\blacktriangleright} t_2$$

は項 $t_1$ が一回の $\blacktriangleleft\blacktriangleright$ 簡約で $t_2$ になることを表す．

$$t_1 \longrightarrow_\Lambda t_2$$

は項 $t_1$ が一回の $\Lambda$ 簡約で $t_2$ になることを表す．

$$\Gamma \vdash K_1 \equiv K_2@A$$

はカインド $K_1$ と $K_2$ が環境 $\Gamma$ の下でステージ $A$ において等価であることを表す．

$$\Gamma \vdash T_1 \equiv T_2@A$$

は型 $T_1$ と $T_2$ が環境 $\Gamma$ の下でステージ $A$ において等価であることを表す．

$$\Gamma \vdash t_1 \equiv t_2@A$$

は項 $T_1$ と $T_2$ が環境 $\Gamma$ の下でステージ $A$ において等価であることを表す．

### 2.3 簡約規則

$\lambda^{\mathrm{MD}}$ には 3 種類の簡約規則がある．

$\longrightarrow_\beta$ は $\beta$ 簡約である．$\longrightarrow_\beta$ は

$$(\lambda x : T.t_1)\ t_2 \longrightarrow_\beta t_1[x \mapsto t_2]$$

を満たす項に関する最小の合同な二項関係である．ここで $t_1[x \mapsto t_2]$ は $t_1$ 中の $x$ の自由な出現に $t_2$ を代入して得られる項である．ただし $T$ の中に含まれる項については $\beta$ 簡約を行わないものとする．

$\longrightarrow_{\blacktriangleleft\blacktriangleright}$ はコード化とエスケープに関する簡約規則である．$\longrightarrow_{\blacktriangleleft\blacktriangleright}$ は

$$\blacktriangleleft_\alpha\ (\blacktriangleright_\alpha\ t)\ \longrightarrow_{\blacktriangleleft\blacktriangleright} t$$

を満たす項に関する最小の合同な二項関係である．この規則は項 $t$ をコード化した $\blacktriangleright_\alpha\ t$ を escape すると $t$ に戻ることを表している．この簡約も依存型のパラメータの項については行わないものとする．

$\longrightarrow_\Lambda$ はステージ変数の書き換え規則である．$\longrightarrow_\Lambda$

は

$$(\Lambda\alpha.t)\epsilon \longrightarrow_\Lambda t[\alpha \mapsto \epsilon]$$

を満たす項に関する最小の二項関係である．ここで $t[\alpha \mapsto \epsilon]$ は $t$ 中の $\alpha$ をすべて $\epsilon$ で置き換えたものである．$\blacktriangleright_\alpha$，$\blacktriangleleft_\alpha$，$\%_\alpha$ の $\alpha$ に $\epsilon$ を代入するとこれらの 3 つの記号は消える．Tsukada, Igarashi [13] や Hanada, Igarashi [6] で議論されているように，ステージ変数に対する $\epsilon$ の代入はコードの実行に対応している．この簡約も $\longrightarrow_\beta$ や $\longrightarrow_{\blacktriangleleft\blacktriangleright}$ と同様に依存型のパラメータの項については行わないものとする．

### 2.4 型システム

$\lambda^{\mathrm{MD}}$ の型システムについて述べる．$\lambda^{\mathrm{MD}}$ の型システムは型付け規則，カインド付け規則，カインドの適格性判断規則を定義する部分と項，型，カインドの等価性を定義する部分からなる．

#### 2.4.1 型付け規則，カインド付け規則，カインドの適格性判断規則

図 1 に $\lambda^{\mathrm{MD}}$ の型付け規則，カインド付け規則，カインドの適格性判断規則を示す．

まず型付け規則について説明する．T-Var, T-Abs, T-App は単純型付きラムダ計算と同じである．T-$\blacktriangleright$, T-$\blacktriangleleft$, T-Gen, T-Ins, T-Csp は $\lambda^{\triangleright\%}$ 由来の規則である．T-$\blacktriangleright$, T-$\blacktriangleleft$, T-Gen, T-Ins はそれぞれ，プログラムのコード化，escape，ステージ変数の抽象，ステージの適用のための型付け規則である．T-Csp は CSP のための型付け規則であり，前のステージにおける項を次のステージの項として扱い，その項の型を変えずに次のステージでの型としている．

T-Conv は依存型に関する規則であり，ある項 $t$ に型 $T$ がつきさらに $T$ が $T'$ と等価ならば，$t$ に $T'$ を付けてもよいという規則である．T-Conv は例えば図 2 のように使われる．この例は $\lambda^{\mathrm{MD}}$ に含まれない自然数に関する演算を含むため，厳密なものではない．しかし T-Conv の直感的な意味をよく捉えている．この導出木はある項 $t$ に長さ $(3+4)$ の配列型がつき，さらに長さ $(3+4)$ の配列型と 7 の配列型が等価なので，$t$ に長さ 7 の配列型をつけられるということを表している．

次にカインド付け規則について説明する．K-Var, K-Abs，K-App，K-Conv は $\lambda$LF 由来の規則である．K-$\triangleright$, K-Gen は $\lambda^{\mathrm{MD}}$ で新たに加わった規則である．K-$\triangleright$ は構文規則の中の $\triangleright_\alpha T$ に対応し，K-Gen は $\forall\alpha.T$ に対応している．

K-Csp は型の CSP に関するカインド付け規則であり，$\lambda^{\mathrm{MD}}$ に特有の規則である．

$$\frac{\Gamma \vdash T :: K@A}{\Gamma \vdash T :: K@A\alpha}\text{K-Csp}$$

K-Csp はステージ $A$ で型 $T$ に $K$ というカインドがつくのなら，ステージ $A\alpha$ でも型 $T$ に $K$ を導出できるということを意味している．例えば，ステージ $\epsilon$ で定義した型変数 Vect はステージ $\alpha$ でも，ステージ $\beta$ でも利用可能である．

最後にカインドの適格性判断規則について説明する．W-Star, W-Abs は構文規則の中の $*$ と $\Pi x : T.K$ に対応するカインドの適格性判断規則である．W-$\triangleright$, W-Csp はそれぞれ K-$\triangleright$, K-Csp に対応するカインドの適格性判断規則である．

#### 2.4.2 等価性規則

型付け規則に加えて $\lambda^{\mathrm{MD}}$ には項等価性，型等価性，カインド等価性を定義する規則がある．図 3，図 4，図 5 に $\lambda^{\mathrm{MD}}$ のカインド等価性，型等価性，項等価性規則を示す．

型等価性の規則は T-Conv が等価な型の付け替えを許すために必要である．図 4 の QT-で始まるものが型等価性の規則である．また依存型においては型の中に項が含まれる場合があるため，項等価性の規則も必要である．図 5 の Q-で始まるものは項の等価性規則である．さらに QT-App の中にカインド同士の等価性判定が含まれるので，カインドの等価性規則も必要である．図 3 の QK-で始まるものがカインドの等価性規則である．

型等価性の規則とは型上の関係 $\equiv$ を定義する規則であり，3 種類に分けられる．1 種類目は $\equiv$ を型に関する合同関係にするための規則であり，QT-Abs, QT-App，QT-$\triangleright$，QT-Gen，QT-Csp が該当する．2 種類目は $\equiv$ を同値関係にするための規則であり，QT-Refl，QT-Sym，QT-Trans が該当する．

項等価性の規則は項上の関係 $\equiv$ を定義する規則であり，型等価性の規則と同様に 3 種類に分けられ

$\boxed{\Gamma \vdash K \text{ kind}}$ **Well-formed kinds**

$$\frac{}{\Gamma \vdash * \text{ kind}@A}\text{W-Star} \qquad \frac{\Gamma \vdash T :: *@A \qquad \Gamma, x : T@A \vdash K \text{ kind}@A}{\Gamma \vdash (\Pi x : T.K) \text{ kind}@A}\text{W-Abs}$$

$$\frac{\Gamma \vdash K \text{ kind}@A}{\Gamma \vdash K \text{ kind}@A\alpha}\text{W-Csp} \qquad \frac{\Gamma \vdash K \text{ kind}@A\alpha}{\Gamma \vdash K \text{ kind}@A}\text{W-}\triangleright$$

$\boxed{\Gamma \vdash T :: K}$ **Kinding**

$$\frac{\Gamma \vdash K \text{ kind}@A}{\Gamma, X :: K@A, \Delta \vdash X :: K@A}\text{K-Var} \qquad \frac{\Gamma \vdash T_1 :: *@A \qquad \Gamma, x : T_1@A \vdash T_2 :: K_2@A}{\Gamma \vdash (\Pi x : T_1.T_2) :: (\Pi x : T_1.K_2)@A}\text{K-Abs}$$

$$\frac{\Gamma \vdash T_2 :: (\Pi x : T_1.K)@A \qquad \Gamma \vdash t_1 : T_1@A}{\Gamma \vdash T_2\ t_1 :: K[x \mapsto t_1]@A}\text{K-App} \qquad \frac{\Gamma \vdash T :: K@A \qquad \Gamma \vdash K \equiv K' \text{ kind}@A}{\Gamma \vdash T :: K'@A}\text{K-Conv}$$

$$\frac{\Gamma \vdash T :: K@A\alpha}{\Gamma \vdash \triangleright_\alpha T :: K@A}\text{K-}\triangleright \qquad \frac{\Gamma \vdash T :: K@A \qquad \alpha \notin \text{FTV}(\Gamma) \cup \text{FTV}(\text{A})}{\Gamma \vdash \forall\alpha.T :: K@A}\text{K-Gen} \qquad \frac{\Gamma \vdash T :: K@A}{\Gamma \vdash T :: K@A\alpha}\text{K-Csp}$$

$\boxed{\Gamma \vdash t : T}$ **Typing**

$$\frac{\Gamma \vdash T :: *@A}{\Gamma, x : T@A, \Delta \vdash x : T@A}\text{T-Var} \qquad \frac{\Gamma \vdash T_2 :: *@A \qquad \Gamma, x : T_2@A \vdash t_1 : T_1@A}{\Gamma \vdash (\lambda x : T_2.t_1) : (\Pi x : T_2.T_1)@A}\text{T-Abs}$$

$$\frac{\Gamma \vdash t_1 : (\Pi x : T_2.T_1)@A \qquad \Gamma \vdash t_2 : T_2@A}{\Gamma \vdash t_1\ t_2 : T_1[x \mapsto t_2]@A}\text{T-App} \qquad \frac{\Gamma \vdash t : T@A \qquad \Gamma \vdash T \equiv T' :: *@A}{\Gamma \vdash t : T'@A}\text{T-Conv}$$

$$\frac{\Gamma \vdash t : T@A\alpha}{\Gamma \vdash \blacktriangleright_\alpha\ t : \triangleright_\alpha T@A}\text{T-}\blacktriangleright \qquad \frac{\Gamma \vdash t : \triangleright_\alpha T@A}{\Gamma \vdash \blacktriangleleft_\alpha\ t : T@A\alpha}\text{T-}\blacktriangleleft \qquad \frac{\Gamma \vdash t : T@A \qquad \alpha \notin \text{FTV}(\Gamma) \cup \text{FTV}(\text{A})}{\Gamma \vdash \Lambda\alpha.t : \forall\alpha.T@A}\text{T-Gen}$$

$$\frac{\Gamma \vdash t : \forall\alpha.T@A}{\Gamma \vdash t\ \epsilon : T[\alpha \mapsto \epsilon]@A}\text{T-Ins} \qquad \frac{\Gamma \vdash t : T@A}{\Gamma \vdash \%_\alpha t : T@A\alpha}\text{T-Csp}$$

ただし

FTV(Γ) は環境 Γ 中の，FTV($A$) はステージ $A$ 中の自由なステージ変数の集合を表す．

$\Gamma \vdash T_1 \equiv T_2 :: K@A$ は $\Gamma \vdash T_1 \equiv T_2@A$ かつ $\Gamma \vdash T_1 :: K@A$ かつ $\Gamma \vdash T_2 :: K@A$ の略記であり，

$\Gamma \vdash K_1 \equiv K_2 \text{ kind}@A$ は $\Gamma \vdash K_1 \equiv K_2@A$ かつ $\Gamma \vdash K_1 \text{ kind}@A$ かつ $\Gamma \vdash K_2 \text{ kind}@A$ の略記である．

**図 1　$\lambda^{\mathbf{MD}}$ の型付け規則，カインド付け規則，カインドの適格性判断規則**

$$\frac{\Gamma \vdash t : \text{Vect } (3+4)@\epsilon \qquad \Gamma \vdash \text{Vect}(3+4) \equiv \text{Vect } 7@\epsilon}{\Gamma \vdash t : \text{Vect } 7@\epsilon}\ \text{T-Conv}$$

**図 2　T-Conv の使用例**

る．Q-Abs，Q-App，Q-▶，Q-◀，Q-Gen，Q-Ins，Q-Csp は ≡ を項に関する合同関係にするための規則である．同値関係に関する規則は Q-Refl，Q-Sym，Q-Trans である．簡約規則に関する規則は Q-$\beta$，Q-◀▶，Q-Λ，Q-%である．このうち Q-$\beta$，Q-◀▶，Q-Λ はそれぞれ $\longrightarrow_\beta$，$\longrightarrow_{\blacktriangleleft\blacktriangleright}$，$\longrightarrow_\Lambda$ に対応するものであり，簡約される前の項と簡約された後の項の組を等価性に入れる規則である．Q-Csp は CSP を表す記号% を扱うために導入された規則である．この規則の意味は，ある項 $t_1$ がステージ $A$ と $A\alpha$ の両方で導出されたならば，$\%_\alpha t_1$ と $t_1$ を等価性に加えるというものである．$\lambda^{\text{MD}}$ ではステージ変数に $\epsilon$ 以外を代入することはないことから，この規則の正当性が確認できる．つまりすべてのコード化されたプログラ

$\boxed{\Gamma \vdash K_1 \equiv K_2 @ A}$ **Kind Equivalence**

$\equiv$ の合同性のための規則

$$\frac{\Gamma \vdash T_1 \equiv T_2 :: * @ A \qquad \Gamma, x : T_1 @ A \vdash K_1 \equiv K_2 @ A}{\Gamma \vdash \Pi x : T_1 . K_1 \equiv \Pi x : T_2 . K_2 @ A}\text{QK-Abs} \qquad \frac{\Gamma \vdash K_1 \equiv K_2 @ A}{\Gamma \vdash K_1 \equiv K_2 @ A\alpha}\text{QK-Csp}$$

$\equiv$ を同値関係にするための規則

$$\frac{\Gamma \vdash K \text{ kind} @ A}{\Gamma \vdash^A K \equiv K}\text{QK-Refl} \qquad \frac{\Gamma \vdash K_1 \equiv K_2 @ A}{\Gamma \vdash K_2 \equiv K_1 @ A}\text{QK-Sym}$$

$$\frac{\Gamma \vdash K_1 \equiv K_2 @ A \qquad \Gamma \vdash K_2 \equiv K_3 @ A}{\Gamma \vdash K_1 \equiv K_3 @ A}\text{QK-Trans}$$

**図 3** $\lambda^{\mathbf{MD}}$ のカインド等価性規則

$\boxed{\Gamma \vdash T_1 \equiv T_2 @ A}$ **Type Equivalence**

$\equiv$ の合同性のための規則

$$\frac{\Gamma \vdash T_1 \equiv T_2 :: * @ A \qquad \Gamma, x : T_1 @ A \vdash T_3 \equiv T_4 @ A}{\Gamma \vdash \Pi x : T_1 . T_3 \equiv \Pi x : T_2 . T_4 @ A}\text{QT-Abs}$$

$$\frac{\Gamma \vdash T_1 \equiv T_2 :: \Pi x : T_3 . K @ A \qquad \Gamma \vdash t_1 \equiv t_2 : T_3 @ A}{\Gamma \vdash T_1\ t_1 \equiv T_2\ t_2 @ A}\text{QT-App}$$

$$\frac{\Gamma \vdash T_1 \equiv T_2 @ A\alpha}{\Gamma \vdash \triangleright_\alpha T_1 \equiv \triangleright_\alpha T_2 @ A}\text{QT-}\triangleright \qquad \frac{\Gamma \vdash T_1 \equiv T_2 @ A \qquad \alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV(A)}}{\Gamma \vdash \forall \alpha . T_1 \equiv \forall \alpha . T_2 @ A}\text{QT-Gen}$$

$$\frac{\Gamma \vdash T_1 \equiv T_2 @ A}{\Gamma \vdash T_1 \equiv T_2 @ A\alpha}\text{QT-Csp}$$

$\equiv$ を同値関係にするための規則

$$\frac{\Gamma \vdash T :: K @ A}{\Gamma \vdash T \equiv T @ A}\text{QT-Refl} \qquad \frac{\Gamma \vdash T_1 \equiv T_2 @ A}{\Gamma \vdash T_2 \equiv T_1 @ A}\text{QT-Sym}$$

$$\frac{\Gamma \vdash T_1 \equiv T_2 @ A \qquad \Gamma \vdash T_2 \equiv T_3 @ A}{\Gamma \vdash T_1 \equiv T_3 @ A}\text{QT-Trans}$$

ただし

$\Gamma \vdash t_1 \equiv t_2 : T @ A$ は $\Gamma \vdash t_1 \equiv t_2 @ A$ かつ $\Gamma \vdash t_1 : T$ かつ $\Gamma \vdash t_2 : T$ の略記であり，

$\Gamma \vdash T_1 \equiv T_2 :: K @ A$ は $\Gamma \vdash T_1 \equiv T_2 @ A$ かつ $\Gamma \vdash T_1 :: K$ かつ $\Gamma \vdash T_2 :: K$ の略記である．

**図 4** $\lambda^{\mathbf{MD}}$ の型等価性規則

ムが実行されたあとでは，すべてのステージ変数に $\epsilon$ が代入され%はすべて消滅するので，$\%_\alpha t_1$ と $t_1$ は全く同じ項になる．

カインド等価性はカインド上の関係 $\equiv$ を定義する規則であり，2 種類に分けられる．1 種類目は $\equiv$ をカインドに関する合同関係にするための規則であり，QK-Abs，QK-Cps が該当する．2 種類目は $\equiv$ を同値関係にするための規則であり，QK-Refl，QK-Sym，QK-Trans が該当する．

これらの等価性の規則は，単純型付きラムダ計算の

$\boxed{\Gamma \vdash t_1 \equiv t_2 @ A}$ **Term Equivalence**

$\equiv$ の合同性のための規則

$$\frac{\Gamma \vdash S_1 \equiv S_2 @ A \qquad \Gamma, x : S_1 @ A \vdash t_1 \equiv t_2 @ A}{\Gamma \vdash \lambda x : S_1 . t_1 \equiv \lambda x : S_2 . t_2 @ A}\text{Q-Abs}$$

$$\frac{\Gamma \vdash t_1 \equiv s_1 : (\Pi x : T_2 . T_1) @ A \qquad \Gamma \vdash t_2 \equiv s_2 : T_2 @ A}{\Gamma \vdash t_1\ t_2 \equiv s_1\ s_2 @ A}\text{Q-App}$$

$$\frac{\Gamma \vdash t_1 \equiv t_2 @ A\alpha}{\Gamma \vdash \blacktriangleright_\alpha\ t_1 \equiv \blacktriangleright_\alpha\ t_2 @ A}\text{Q-}\blacktriangleright \qquad \frac{\Gamma \vdash t_1 \equiv t_2 : \triangleright_\alpha T @ A}{\Gamma \vdash \blacktriangleleft_\alpha\ t_1 \equiv \blacktriangleleft_\alpha\ t_2 @ A\alpha}\text{Q-}\blacktriangleleft$$

$$\frac{\Gamma \vdash t_1 \equiv t_2 @ A \qquad \alpha \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(A)}{\Gamma \vdash \Lambda\alpha . t_1 \equiv \Lambda\alpha . t_2 @ A}\text{Q-Gen} \qquad \frac{\Gamma \vdash t_1 \equiv t_2 : \forall \alpha . T @ A}{\Gamma \vdash t_1\ \epsilon \equiv t_2\ \epsilon @ A}\text{Q-Ins}$$

$$\frac{\Gamma \vdash t_1 \equiv t_2 @ A}{\Gamma \vdash \%_\alpha t_1 \equiv \%_\alpha t_2 @ A\alpha}\text{Q-Csp}$$

$\equiv$ を同値関係にするための規則

$$\frac{}{\Gamma \vdash t \equiv t @ A}\text{Q-Refl} \qquad \frac{\Gamma \vdash t \equiv s @ A}{\Gamma \vdash s \equiv t @ A}\text{Q-Sym} \qquad \frac{\Gamma \vdash t_1 \equiv t_2 @ A \qquad \Gamma \vdash t_2 \equiv t_3 @ A}{\Gamma \vdash t_1 \equiv t_3 @ A}\text{Q-Trans}$$

簡約規則に関する規則

$$\frac{\Gamma, x : S @ A \vdash t : T @ A \qquad \Gamma \vdash s : S @ A}{\Gamma \vdash (\lambda x : S . t)\ s \equiv t[x \mapsto s] @ A}\text{Q-}\beta \qquad \frac{\Gamma \vdash t_1 \equiv t_2 @ A}{\Gamma \vdash \blacktriangleleft_\alpha\ (\blacktriangleright_\alpha\ t_1) \equiv t_2 @ A}\text{Q-}\blacktriangleleft\blacktriangleright$$

$$\frac{\Gamma \vdash \Lambda\alpha . t_1 : \forall \alpha . T_1 @ A}{\Gamma \vdash (\Lambda\alpha . t_1)\ \epsilon \equiv t_1[\alpha \to \epsilon] @ A}\text{Q-}\Lambda \qquad \frac{\Gamma \vdash t_1 : T_1 @ A\alpha \qquad \Gamma \vdash t_1 : T_1' @ A}{\Gamma \vdash \%_\alpha t_1 \equiv t_1 @ A\alpha}\text{Q-}\%$$

ただし

$\Gamma \vdash t_1 \equiv t_2 : T @ A$ は $\Gamma \vdash t_1 \equiv t_2 @ A$ かつ $\Gamma \vdash t_1 : T$ かつ $\Gamma \vdash t_2 : T$ の略記である．

**図 5** $\lambda^{\mathrm{MD}}$ **の項等価性規則**

簡約だけでなく，ステージに関する抽象やその簡約，CSP の記号%を扱えるように定義されている．既存研究にはこのような等価性の規則は存在しないため，これらの規則は本研究の貢献である．

## 3 例

$\lambda^{\mathrm{MD}}$ によって型付けされる例を一つ挙げる．まずわかりやすさのために let 式を使って例を表示したあと，let 式を $\lambda^{\mathrm{MD}}$ に変換して表示し，その型付け導出を表示する．ただしこの節では整数 Int を定義せずに使う．

ラムダ項と適用を let 式に変換したものを図 6 に示す．この例では環境で型変数 Index を定義している．Index は範囲付きの整数を表す．

Index は整数 $i$ を受け取って，範囲を 0 から $i-1$ に制限した整数の型を返す．例えば (Index 13) 型の $x$ は 0 から 12 までの値しか取らない．これは 1.1 節で導入した Index 型と同じものである．

さらに，(Index 13) 型の値 $m$ をステージ $\alpha$ で定義している．プログラム中では型 Index ($\%_\alpha$ 13) を持つ変数 $y$ に $m$ を代入し，その後その $y$ をそのまま返すという処理をコード化している．

let 式を用いずに同じ例を表示したものを図 7 に示し，更にその型付け導出木を図 8 に示す．導出木のサイズが非常に大きいので，導出木を 3 つに分割している．

導出木のうち $\lambda^{\mathrm{MD}}$ に特徴的な部分を説明する．

まず label-$p_1$ で変数 $y$ が型 Index $(\%_\alpha 13)@\alpha$ を持つことを導出している．導出木の左上の部分において，K-Csp で型変数 Index をステージ $\epsilon$ からステージ $\alpha$ に持ち上げて使えるのは $\lambda^{\mathrm{MD}}$ の大きな特徴である．K-Csp がなければ型変数 Index をステージ $\epsilon$ でも定義しておく必要がある．

さらに label-$p_2$ で型 Index $(\%_\alpha 13)$ と型 Index 13 がステージ $\alpha$ で等価であることを導出している．特に左上の部分において，QT-Csp を用いて型 Index が型 Index とステージ $\alpha$ で等価であることを導出しており，これは $\lambda^{\mathrm{MD}}$ で新たに導入した QT-Csp が機能している例である．

## 4 性質と証明

$\lambda^{\mathrm{MD}}$ の満たすべき性質について述べる．本研究では $\lambda^{\mathrm{MD}}$ の強正規化性，合流性と型健全性を証明することを目標としている．

$\lambda^{\mathrm{MD}}$ の強正規化性は λLF と同様に $\lambda^{\mathrm{MD}}$ の強正規化性を単純型付けラムダ計算のそれに帰着させることで示すことができる．

$\lambda^{\mathrm{MD}}$ の合流性は簡約に関する帰納法で弱合流性を示したあと，Newman's Lemma と強正規化性から示せる．

型健全性を示すためには簡約による型の保存と，型のつく項に対する簡約の進行性を示す必要がある．簡約による型の保存は以下の定理を導出に関する帰納法で証明することで示せると考えている．

**定理 1** (代入補題)**.**
$\Gamma, x : T_2@B, \Delta \vdash J@A$ かつ $\Gamma \vdash t_2 : T_2@B$
ならば $\Gamma, \Delta[x \mapsto t_2] \vdash J[x \mapsto t_2]@A$.

ただし $t_1 : T_1@A$, $T_1 :: K_1@A$, $K_1$ kind$@A$, $t_1 \equiv t_2@A$, $T_1 \equiv T_2@A$, $K_1 \equiv K_2@A$ をまとめて $J@A$ と書く．

**定理 2** (整合性)**.**

- $\Gamma \vdash T_1 :: K_1@A$ ならば $\Gamma \vdash K_1$ kind$@A$.
- $\Gamma \vdash t_1 : T_1@A$ ならば $\Gamma \vdash T_1 :: *@A$.
- $\Gamma \vdash K_1 \equiv K_2@A$ かつ $\Gamma \vdash K_1$ kind$@A$ ならば $\Gamma \vdash K_2$ kind$@A$.
- $\Gamma \vdash T_1 \equiv T_2@A$ かつ $\Gamma \vdash T_1 :: K_1@A$ ならば $\Gamma \vdash T_2 :: K_1@A$.
- $\Gamma \vdash t_1 \equiv t_2@A$ かつ $\Gamma \vdash t_1 : T_1@A$ ならば $\Gamma \vdash t_2 : T_1@A$.

**定理 3** (簡約による型の保存)**.**

- $\Gamma \vdash t_1 : T_1@A$ かつ $t_1 \longrightarrow_\beta t_2$ ならば $\Gamma \vdash t_2 : T_1@A$.
- $\Gamma \vdash t_1 : T_1@A$ かつ $t_1 \longrightarrow_{\blacktriangleleft\blacktriangleright} t_2$ ならば $\Gamma \vdash t_2 : T_1@A$.
- $\Gamma \vdash t_1 : T_1@A$ かつ $t_1 \longrightarrow_\Lambda t_2$ ならば $\Gamma \vdash t_2 : T_1@A$.

進行性を示すためには $\lambda^{\mathrm{MD}}$ における値を定義する必要がある．しかし，$\lambda^{\mathrm{MD}}$ は $\lambda^{\triangleright\%}$ の拡張であり，さらに項の構文は $\lambda^{\triangleright\%}$ と同じであるため，$\lambda^{\mathrm{MD}}$ の進行性は $\lambda^{\triangleright\%}$ の進行性に簡単に帰着できる．本研究では $\lambda^{\triangleright\%}$ の値の定義を流用して，$\lambda^{\mathrm{MD}}$ の進行性を示す予定である．

## 5 関連研究

プログラミング言語における依存型の理論は 1980 年代半ばに確立された．この時期の依存型に関する論文として，Meyer と Reinhold による $\lambda^{\Pi}$ [9]，Coquand と Huet による Calculus of Constructions [4]，Harper，Honsell，Plotkin による λLF [7] が挙げられる．本研究では依存型の基礎として Aspinall, Hoffman [1] を使った．

依存型を扱えるプログラミング言語としては Idris [3] が存在する．また，Coq [12] や Agda [2] などの定理証明支援システムでも依存型を使うことができる．

依存型を実際にプログラムの安全性向上のために用いた研究としては Xi と Pfenning によるもの [14] がある．この研究では Standard ML を限定された形の依存型を使えるように拡張し，配列の境界検査の回数を減らすことに成功している．

一方，マルチステージプログラミングの型理論も長く研究されている．Davies は $\lambda^\circ$ [5] でマルチステージプログラミングと様相論理の間にカリーハワード同型があることを明らかにした．しかし，$\lambda^\circ$ にはコードの実行や CSP が存在しなかった．

Taha と Sheard は MetaML [11] でコードの実行と CSP を導入した．さらに Taha と Nielsen は $\lambda^\alpha$

$$\mathrm{Index} :: \Pi x : \mathrm{Int}. * @\epsilon, m : \mathrm{Index}\ 13@\alpha \vdash (\blacktriangleright_\alpha\ (\ \mathrm{let}\ y = m : \mathrm{Index}\ (\%_\alpha\ 13)\ \mathrm{in}\ y)) : \triangleright_\alpha\ \mathrm{Index}\ 13@\epsilon$$

**図 6　見やすさのために let 式を用いて表示した $\lambda^{\mathbf{MD}}$ のプログラムの例**

$$\mathrm{Index} :: \Pi x : \mathrm{Int}. * @\epsilon, m : \mathrm{Index}\ 13@\alpha \vdash \blacktriangleright_\alpha\ ((\lambda(y : \mathrm{Index}\ (\%_\alpha 13)).y)\ m) : \triangleright_\alpha \mathrm{Index}\ 13@\alpha$$

**図 7　let 式を用いずに表示した $\lambda^{\mathbf{MD}}$ のプログラムの例**

$\Gamma$ を $\mathrm{Index} :: \Pi(x : \mathrm{Int}). * @\epsilon, m : \mathrm{Index}\ 13@\alpha$ とする．

$p_1$ を $\Gamma, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha \vdash y : \mathrm{Index}(\%_\alpha 13)@\alpha$ とする．

$p_2$ を $\Gamma, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha \vdash \mathrm{Index}\ (\%_\alpha 13) \equiv \mathrm{Index}\ 13@\alpha$　とする．

$$\dfrac{\dfrac{\dfrac{\dfrac{(\text{goto label-}p_1)}{p_1} \quad \dfrac{(\text{goto label-}p_2)}{p_2}}{\Gamma, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha \vdash y : \mathrm{Index}\ 13@\alpha}\ \text{T-Conv} \quad \dfrac{\vdots}{\Gamma \vdash \mathrm{Index}\ (\%_\alpha 13) : *@\alpha}}{\Gamma \vdash (\lambda(y : \mathrm{Index}\ (\%_\alpha 13)).y) : \Pi(y : \mathrm{Index}\ 13).\mathrm{Index}\ 13@\alpha}\ \text{T-Abs} \quad \dfrac{\vdots}{\Gamma \vdash m : \mathrm{Index}\ 13@\alpha}\ \text{T-App}}{\dfrac{\Gamma \vdash (\lambda(y : \mathrm{Index}\ (\%_\alpha 13)).y)\ m : \mathrm{Index}\ 13@\alpha}{\Gamma \vdash \blacktriangleright_\alpha\ ((\lambda(y : \mathrm{Index}\ (\%_\alpha 13)).y)\ m) : \triangleright_\alpha \mathrm{Index}\ 13@\epsilon}\ \text{T-}\blacktriangleright}$$

label-$p_1$:

$\Gamma'$ を $\mathrm{Index} :: \Pi x : \mathrm{Int}. * @\epsilon, m : \mathrm{Index}\ 13@\alpha, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha$　とする．

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma' \vdash \mathrm{Index} :: \Pi x : \mathrm{Int}. * @\epsilon}}{\Gamma' \vdash \mathrm{Index} :: \Pi x : \mathrm{Int}. * @\alpha}\ \text{K-Csp} \quad \dfrac{\dfrac{\vdots}{\Gamma' \vdash 13 : \mathrm{Int}@\epsilon}}{\Gamma' \vdash (\%_\alpha 13) : \mathrm{Int}@\alpha}\ \text{T-Csp}}{\Gamma' \vdash \mathrm{Index}\ (\%_\alpha 13) :: *@\alpha}\ \text{K-App}}{\Gamma, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha \vdash y : \mathrm{Index}\ (\%_\alpha 13)@\alpha}\ \text{T-Var}$$

label-$p_2$:

$\Gamma'$ を $\mathrm{Index} :: \Pi(x : \mathrm{Int}). * @\epsilon, m : \mathrm{Index}\ 13@\alpha, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha$　とする．

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathrm{Index} :: \Pi x : \mathrm{Int}. * @\epsilon}\ \text{QT-Refl}}{\Gamma \vdash \mathrm{Index} \equiv \mathrm{Index}@\epsilon}\ \text{QT-Csp}}{\Gamma' \vdash \mathrm{Index} \equiv \mathrm{Index}@\alpha} \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash \mathrm{Index} :: \Pi x : \mathrm{Int}. * @\epsilon}}{\Gamma' \vdash \mathrm{Index} :: \Pi x : \mathrm{Int}. * @\alpha}\ \text{K-Csp 略記} \quad \dfrac{\vdots}{\Gamma' \vdash 13 : \mathrm{Int}@\alpha} \quad \dfrac{\vdots}{\Gamma' \vdash 13@\epsilon}\ \text{Q-\%}}{\Gamma' \vdash \mathrm{Index} \equiv \mathrm{Index} :: \Pi x : \mathrm{Int}. * @\alpha} \quad \dfrac{}{\Gamma' \vdash (\%_\alpha 13) \equiv 13 : \mathrm{Int}@\alpha}\ \text{QT-App}}{\Gamma, y : \mathrm{Index}\ (\%_\alpha 13)@\alpha \vdash \mathrm{Index}\ (\%_\alpha 13) \equiv \mathrm{Index}\ 13@\alpha}$$

**図 8　$\lambda^{\mathbf{MD}}$ における型付け導出**

[10] で Environment Classifer を導入し，bracket，escape，run，cross-stage persistence の 4 種類の操作を可能であり，型安全な型システムを構築した．また，Tsukada と Igarashi は $\lambda^{\triangleright}$ [13] で Environment Classifer を含む計算体系に対応する論理体系を発見し，Environment Classifer を $\epsilon$ で置き換えることでコードの実行を表現できることを示した．

Hanada と Igarashi は $\lambda^{\triangleright\%}$ で $\lambda^{\triangleright}$ に CSP を付け加えることに成功した．$\lambda^{\triangleright\%}$ は本研究におけるマルチステージプログラミングの体系の基礎である．

実際に使えるマルチステージプログラミング言語としては BER MetaOCaml [8] が挙げられる．MetaOCaml BER はコードの実行や CSP をサポートしている．

いずれの体系も依存型を扱っておらず，我々の知る限り bracket, escape, run, CSP を持つ依存型システムは本研究が初めてである．

## 6 おわりに

本研究は依存型を含むマルチステージプログラミングの型理論 $\lambda^{\mathrm{MD}}$ を定義した．特にコード化された値をパラメータに持つ型の等価性規則と，CSP を含むマルチステージプログラミング言語におけるカインド付け規則，カインドの適格性判断規則を与えた．

今後は定義した型システムと簡約規則のもとで，強正規化性と型健全性が保たれることを示す．現在，この証明に着手しており，既存研究と同一の手法で証明する予定である．

さらに $\lambda^{\mathrm{MD}}$ を実際に使うために，$\lambda^{\mathrm{MD}}$ を基礎とした型システムを設計し，そのインタプリタや JIT コンパイラの作成を予定している．インタプリタだけでなく JIT コンパイラを作成する理由は，マルチステージプログラミングの利点が活かされる可能性があると考えているからである．

## 謝辞

原稿を読み，有益なコメントを寄せてくれた馬谷誠二氏に謝意を表したい．本研究は JSPS 科研費 17H01723 の助成を受けたものである．

## 参 考 文 献

[1] Aspinall, D. and Hofmann, M.: *Advanced topics in types and programming languages*, MIT press, 2005, chapter 2 Dependent Types.

[2] Bove, A., Dybjer, P., and Norell, U.: A Brief Overview of Agda — A Functional Language with Dependent Types, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, Berlin, Heidelberg, Springer-Verlag, 2009, pp. 73–78.

[3] Brady, E.: Idris, a general-purpose dependently typed programming language: Design and implementation, *Journal of Functional Programming*, Vol. 23, No. 5(2013), pp. 552–593.

[4] Coquand, T. and Huet, G.: The calculus of constructions, Technical Report RR-0530, INRIA, May 1986.

[5] Davies, R.: A temporal-logic approach to binding-time analysis, *Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on*, IEEE, 1996, pp. 184–195.

[6] Hanada, Y. and Igarashi, A.: On cross-stage persistence in multi-stage programming, *International Symposium on Functional and Logic Programming*, Springer, 2014, pp. 103–118.

[7] Harper, R., Honsell, F., and Plotkin, G.: A framework for defining logics, *Journal of the ACM (JACM)*, Vol. 40, No. 1(1993), pp. 143–184.

[8] Kiselyov, O.: The Design and Implementation of BER MetaOCaml - System Description, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, Codish, M. and Sumii, E.(eds.), Lecture Notes in Computer Science, Vol. 8475, Springer, 2014, pp. 86–102.

[9] Meyer, A. R. and Reinhold, M. B.: "Type" is Not a Type, *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '86, New York, NY, USA, ACM, 1986, pp. 287–295.

[10] Taha, W. and Nielsen, M. F.: Environment classifiers, *ACM SIGPLAN Notices*, Vol. 38, No. 1, ACM, 2003, pp. 26–37.

[11] Taha, W. and Sheard, T.: MetaML and Multi-stage Programming with Explicit Annotations, *Theor. Comput. Sci.*, Vol. 248, No. 1-2(2000), pp. 211–242.

[12] The Coq Development Team: The Coq Proof Assistant Reference Manual, 2009.

[13] Tsukada, T. and Igarashi, A.: A Logical Foundation for Environment Classifiers, *Logical Methods in Computer Science*, Vol. Volume 6, Issue 4(2010).

[14] Xi, H. and Pfenning, F.: Eliminating Array Bound Checking Through Dependent Types, *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI '98, New York, NY, USA, ACM, 1998, pp. 249–257.