動的変数をもつ依存型付きラムダ計算

叢悠悠, 浅井健一

お茶の水女子大学
{so.yuyu, asai}@is.ocha.ac.jp

概 要　依存型付き言語では、型を使ってデータの性質を正確に表現することで、信頼性の高いプログラムを実装することができる。こうした言語の表現力を高めるために、必要な制限を設けながら、依存型とさまざまな副作用を組み合わせるという試みがなされてきた。本研究では、動的変数を含む依存型付きラムダ計算を考える。動的変数は、実行時の文脈によって値が定まるという性質をもつが、このような変数が型の中に現れると、その型全体の意味が静的に定まらなくなってしまう。そこで、本研究では、動的変数への参照を一種のエフェクトとみなし、副作用をもつ依存型付き言語と同様に、型の依存性をエフェクトのない項に限定する。また、この制限を利用しながら、選択的な環境渡し形式への変換を定義することによって、プログラムの型を保持したまま動的変数を除去することを可能にする。

1　はじめに

項への依存を許す型「依存型」は、プログラムの信頼性向上に欠かせないツールである。たとえば、「0 以外の整数」「長さが n の配列 」といった型を使うことで、0 による除算や、配列の範囲を超えた参照などのエラーを、型検査時に検知することができる。また、依存型によって表された式を命題、その型をもつ項を証明とみなすことで、あるプログラムが特定の性質を満たすかどうかを議論することもできる。このように、型を用いて安全性を保証するという考え方を実世界のソフトウェアに取り入れるために、多くの研究者が依存型付き言語の拡張を試みている。

その 1 つのアプローチとして、筆者ら [5] は過去に、限定継続命令 `shift`/`reset` [10] をもつ依存型付き言語を設計している。既存研究 [18, 26] が示しているように、依存型と制御演算子を共存させるためには、型の依存性を純粋な項（継続へのアクセスを伴わない項）に制限する必要があるが、`shift`/`reset` を扱う場合も、これらを除去する CPS 変換を定義する際に、同様の制限が必要となった。

一方で、筆者らの先行研究には 2 つの課題が残されている。1 つ目は、継続を捕捉する `shift` と、限定する `reset` の対応関係をユーザが指定できないという点である。`shift` と `reset` は、例外や非決定性など、幅広い副作用を表現できるが [12]、1 つのプログラムの中で複数の副作用を扱うと、ある副作用の演算を表す `shift` が、別の副作用を処理する `reset` と結びついてしまうことがある。このような意図しない結合を避けるためには、`shift` と `reset` にプロンプトとよばれるタグをもたせる必要がある。2 つ目の問題は、CPS 変換後の言語が非叙述性を要求するという点である。非叙述性は、多相型 $\forall\alpha.\, A$ において、α を自分自身（つまり、$\forall\alpha.\, A$）に具体化することを許すが、この性質はさまざまなパラドックスを引き起こす要因となるため [6, 14]、Coq [36] や Agda [29] などの依存型付き言語では、非叙述的な多相型の使用を制限している。したがって、非叙述性への依存は、既存の言語を `shift`/`reset` で拡張することの妨げとなる。

本研究では、これらの問題を解決するための 1 ステップとして、動的変数をもつ依存型付き言語を設計し、それに対する選択的なプログラム変換を定義する（図 1）。動的変数は、実行時の文脈によって値が定まる変数のことであり、その意味論は、プロンプトタグ付き限定継続命令の意味論と深く関わっている [15, 24, 11]。プログラムに含まれる動的変数は、環境渡し形式（Environment-Passing

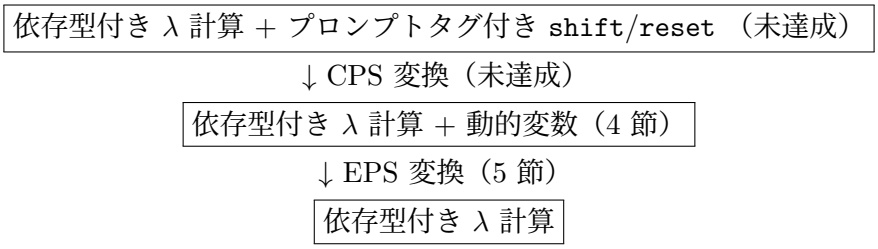

図 1. 最終的な目標と本研究の貢献

項	M	::=	V | p | M N | dlet p = V : A in M
値	V	::=	c | x | λx : A. M
型	A	::=	C | A → B Σ
型環境	Γ	::=	• | Γ, x : A
動的変数の環境	Σ	::=	• | Σ, p : A

図 2. DB の構文

Style; EPS）への変換を通して除去することができるが [27]、本研究ではこれを選択的な変換として定義することで、非叙述性への依存を回避する。

本稿ではまず、動的変数をもつ単純型付き λ 計算を定義し（2 節）、それに対する選択的 EPS 変換を与える（3 節）。次に、言語を依存型で拡張し（4 節）、型の依存性を適切に制限したうえで、EPS 変換を定義する（5 節）。最後に、設計した言語の応用例を紹介し（6 節）、関連研究について議論する（7 節）。

2 単純型付き言語 DB

本節では、動的変数をもつ単純型付き言語 DB を設計する。DB では、動的変数への参照を一種のエフェクトとみなし、実行時のこのようなエフェクトが伴う項と、そうでない項（純粋な項）を明確に区別する。こうすることで、3 節で扱う EPS 変換を選択的な変換として定義できるほか、4 節で設計する依存型付き言語において、型の依存性を適切に制限することが可能となる。

2.1 動的変数の振る舞い

言語の仕様に入る前に、動的変数がどのように振る舞うかを見てみよう。以下のプログラムにおいて、let は静的変数、dlet は動的変数を束縛するものとする。

(1) $\mathsf{let}\ x = 1\ \mathsf{in}\ (\mathsf{let}\ x = 2\ \mathsf{in}\ \lambda y.\, x)\ 0 \triangleright (\mathsf{let}\ x = 2\ \mathsf{in}\ (\lambda y.\, x))\ 0 \triangleright (\lambda y.\, 2)\ 0 \triangleright 2$

(2) $\mathsf{dlet}\ x = 1\ \mathsf{in}\ (\mathsf{dlet}\ x = 2\ \mathsf{in}\ \lambda y.\, x)\ 0 \triangleright \mathsf{dlet}\ x = 1\ \mathsf{in}\ (\lambda y.\, x)\ 0 \triangleright \mathsf{dlet}\ x = 1\ \mathsf{in}\ x \triangleright 1$

静的変数の値は、プログラムの表層的な構造によって定まる。そのため、(1) において、λ 抽象の本体 x は直近の束縛 $x = 2$ によって 2 に置き換わっている。これに対し、動的変数の値は、実行時の文脈によって定まる。先ほど注目した変数 x が実行されるのは、関数 $\lambda y.\, x$ に引数が渡されたあとである。そのため、(2) では $x = 2$ という束縛が無視され、β 簡約後に x を取り囲んでいる束縛 $x = 1$ が使われる。

評価文脈

$$\mathsf{E}[\,] \quad ::= \quad [\,] \mid \mathsf{E}[[\,]\ \mathsf{M}] \mid \mathsf{E}[\mathsf{V}\ [\,]] \mid \mathsf{E}[\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ [\,]]$$

束縛された動的変数

$$\begin{array}{lcl}
\mathrm{BP}\,([\,]) & = & \{\} \\
\mathrm{BP}\,(\mathsf{E}[[\,]\ \mathsf{M}]) & = & \mathrm{BP}\,(\mathsf{E}) \\
\mathrm{BP}\,(\mathsf{E}[\mathsf{V}\ [\,]]) & = & \mathrm{BP}\,(\mathsf{E}) \\
\mathrm{BP}\,(\mathsf{E}[\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ [\,]]) & = & \mathrm{BP}\,(\mathsf{E}) \cup \{\mathsf{p}\}
\end{array}$$

簡約規則

$$\begin{array}{lcl}
(\lambda\,\mathsf{x} : \mathsf{A}.\,\mathsf{M})\ \mathsf{V} & \triangleright & \mathsf{M}[\mathsf{V}/\mathsf{x}] \\
\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ \mathsf{E}[\mathsf{p}] & \triangleright & \mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ \mathsf{E}[\mathsf{V}] \quad \text{if } \mathsf{p} \notin \mathrm{BP}\,(\mathsf{E}) \\
\mathsf{dlet}\ \mathsf{p} = \mathsf{V}_0 : \mathsf{A}\ \mathsf{in}\ \mathsf{V}_1 & \triangleright & \mathsf{V}_1
\end{array}$$

図 3. DB の評価規則

2.2 DB の構文

動的変数の振る舞いが分かったところで、DB の構文（図 2）を見てみよう。DB の項は値と計算からなり、前者は定数 c、静的変数 x、静的変数を束縛する λ 抽象 $\lambda\,\mathsf{x} : \mathsf{A}.\,\mathsf{M}$ を、後者は動的変数 p、関数適用 M N、動的変数を束縛する let 文 $\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ \mathsf{M}$ を含む。束縛変数に型の注釈を付けているのは、構文上等しい 2 つの項が異なる項に EPS 変換されるのを防ぐためである（3.2 節参照）。また、動的変数は値にのみ束縛されるが、限定継続命令を除去する過程において、動的変数はメタ継続の表現に使用されることから、計算への束縛を許す必要はない。

型は定数型 C と関数型 $\mathsf{A} \to \mathsf{B}\ \Sigma$ からなる。後者に現れる Σ は、関数本体から参照される動的変数の型情報が入った環境である。静的変数の型情報は、これと区別して Γ と表す。

2.3 DB の評価規則

次に、DB の評価文脈と簡約規則を定義する（図 3）。これらの規則は、項の実行順序を「値呼びかつ左から右」に規定している。dlet 文に対する 2 つの簡約規則を見ると、この構文によって導入される束縛 $\mathsf{p} = \mathsf{V}$ が有効であるのは、本体に現れる p が実行されるとき（つまり、本体が $\mathsf{E}[\mathsf{p}]$ という形となったとき）のみであることが分かる。これは、本体中の x をすべて V で置き換える静的な let 文の規則 $\mathsf{let}\ x = V\ \mathsf{in}\ M \triangleright M[V/x]$ と異なる。本稿では以下、$\triangleright$ の左辺を簡約基とよび、メタ変数 R で表す。また、図 3 の規則の compatible な閉包を $\mathsf{M} \triangleright \mathsf{N}$、$\mathsf{M} \triangleright \mathsf{N}$ の反射的・推移的閉包を $\mathsf{M} \triangleright^{*} \mathsf{N}$ と表す。

2.4 型システム

続いて、DB の型規則を定義する。本研究では、Rompf ら [31] の `shift`/`reset` 付き λ 計算にならって、項がもつエフェクトの情報を過不足なく伝達するような規則を与える。具体的には、型判断の構成要素として動的変数の環境 Σ を付け加え、この環境に、実行時に参照されるすべての動的変数（かつそれらのみ）をもたせる。

図 4 の規則を見てみよう。まず、定数 c はエフェクトを伴わないため、結論部の Σ が空の環境となっている。ここで、前提部に現れる Φ は定数の型宣言の列である。静的変数 x も類似した規

項の導出規則

$$\frac{c : C \in \Phi}{\Gamma \vdash_\bullet c : C}\ \langle \textsc{Const} \rangle \qquad \frac{x : A \in \Gamma}{\Gamma \vdash_\bullet x : A}\ \langle \textsc{Var} \rangle \qquad \frac{}{\Gamma \vdash_{(\bullet,\, p:A)} p : A}\ \langle \textsc{DVar} \rangle$$

$$\frac{\Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\bullet \lambda x : A.\, M : A \to B\ \Sigma}\ \langle \textsc{Abs} \rangle \qquad \frac{\Gamma \vdash_{\Sigma_0} M : A \to B\ \Sigma_2 \quad \Gamma \vdash_{\Sigma_1} N : A \quad \mathsf{consistent}\,(\Sigma_0 \cup \Sigma_1 \cup \Sigma_2)}{\Gamma \vdash_{(\Sigma_0 \cup \Sigma_1 \cup \Sigma_2)} M\ N : B}\ \langle \textsc{App} \rangle$$

$$\frac{\Gamma \vdash_\bullet V : A \quad \Gamma \vdash_\Sigma M : B \quad \text{if } p : A' \in \Sigma,\, A' = A}{\Gamma \vdash_{(\Sigma \setminus \{p\})} \mathsf{dlet}\ p = V : A\ \mathsf{in}\ M : B}\ \langle \textsc{DLet} \rangle$$

図 4. DB の型規則

則によって導出される。これに対して、動的変数 p は、自身の型情報のみを含む環境 $\bullet, p : A$ で注釈された型判断をもつ。関数 $\lambda x : A.\, M$ は、本体が動的変数を含む場合も、純粋な項としてみなされる。なぜなら、本体に現れる動的変数への参照は、関数が呼び出されたときにはじめて実行されるからである。一方、導出された関数の型には、Σ という注釈が付いている。この注釈は、関数適用のエフェクトを計算する目的で使われる。関数適用の規則を見ると、関数 M、引数 N、関数の本体から参照される動的変数の環境が、それぞれ $\Sigma_0, \Sigma_1, \Sigma_2$ と表されている。これらの動的変数は、関数適用全体の実行時にすべて参照されるため、結論部に置かれる動的変数の環境は、3 つの環境の和をとったものになっている。また、$\mathsf{consistent}\,(\Sigma_0 \cup \Sigma_1 \cup \Sigma_2)$ は、これらの環境に矛盾した型情報（たとえば、$p : \mathsf{int} \in \Sigma_0, p : \mathsf{bool} \in \Sigma_1$）が含まれないことを保証している。dlet 文の規則は、本体の型付けに Σ という環境が使われていた場合、全体の型付けには Σ から p を差し引いた環境が使われることを示している。この規則から、dlet は「p への参照というエフェクトを閉じる」という役割をもつことが分かる。なお、$\langle \textsc{DVar} \rangle, \langle \textsc{App} \rangle, \langle \textsc{DLet} \rangle$ の定義より、動的変数の環境に同じ名前が 2 回以上出現することはない。

既存の型システムとの比較　Kiselyov ら [24] の型システムでは、動的変数の型情報が大域的な環境 Σ に格納されており、すべての項がこの環境のもとで型付けされる。そのため、値の型判断にも空でない Σ が使われる。これは、Kiselyov らの型システムを採用すると、選択的な EPS 変換を定義するための情報が型判断から得られないこと、および、言語を依存型で拡張した際に、型規則によって依存性を制限できないことを意味する。

また、動的変数の情報を大域的な環境として保持する場合、関数型の定義域にエフェクトの情報をもたせる必要がなくなるが、これによって、停止しない項に型を付けることが可能になる。たとえば、$\mathsf{dlet}\ p = \lambda x.\, p\ x\ \mathsf{in}\ p\ 1$ は無限ループを起こすが、Kiselyov の体系では、$p : \mathsf{int} \to \mathsf{int}$ とすると、int 型の項と判断される。一方、DB で同じ項を定義しようとすると、p の型は $\mathsf{int} \to \mathsf{int}\ (\bullet, p : \mathsf{int} \to \mathsf{int}\ (\bullet, p : \ldots))$ となる。そのため、DB では動的変数を利用して無限ループを起こすことはできない。

なお、査読者からの指摘があった通り、関数型に動的変数の環境をもたせると、map などの高階関数を定義する際に、引数として受け取る関数がもつエフェクトを固定する必要がある。これは、DB を限定継続命令を除去する目的で使う上では、問題にならないと考えられる。なぜなら、参照される動的変数の集合は、どのプロンプトタグをもつ `shift` 命令が実行されるかを調べることで定められるからである。

2.5 健全性

DB の型システムは健全である。本稿では、Wright and Felleisen [38] に従って、型保存（preservation）と進行（progress）を証明することで、健全性を示す。

型保存定理は、3 つの補題を使って示される。なお、2 つ目と 3 つ目の補題に現れる型判断 $\Gamma \vdash E[] : A \to A' \; \Sigma_1$ は、評価文脈 $E[]$ の hole が A 型の値に置き換えられたときに、Σ_1 中の動的変数を参照する A' 型の項が得られることを意味する（付録参照）。

補題 1 (代入)**.** $\Gamma, x : A, \Gamma' \vdash_\Sigma M : B$ かつ $\Gamma \vdash_\bullet V : A$ ならば、$\Gamma, \Gamma' \vdash_\Sigma M[V/x] : B$。

補題 2 (分解)**.** $\Gamma \vdash_\Sigma E[M] : A'$ ならば、ある A, Σ_0, Σ_1 に対して、$\Gamma \vdash_{\Sigma_0} M : A$ および $\Gamma \vdash E[] : A \to A' \; \Sigma_1$ が成り立つ。ただし、$(\Sigma_0 \setminus \mathrm{BP}(E[])) \cup \Sigma_1$ は Σ と等しい。

補題 3 (合成)**.** $\Gamma \vdash_{\Sigma_0} M : A$ かつ $\Gamma \vdash E[] : A \to A' \; \Sigma_1$ ならば、$\Gamma \vdash_{((\Sigma_0 \setminus \mathrm{BP}(E[])) \cup \Sigma_1)} E[M] : A'$。

定理 1 (型保存)**.** $\Gamma \vdash_\Sigma M : A$ かつ $M \triangleright^* M'$ ならば、$\Gamma \vdash_\Sigma M' : A$。

Proof. まず、M の導出に関する帰納法によって、1 ステップの簡約が型を保存することを示す。

ケース 1 ($\langle$App$\rangle$)**.**

サブケース 1 ($(\lambda x : A.\, M)\; V \triangleright M[V/x]$)**.** 示したいのは、$\Gamma \vdash_{\Sigma_2} M[V/x] : B$ が導出できることである。$\langle$Abs$\rangle$ より、M は $\Gamma, x : A \vdash_{\Sigma_2} M : B$ という形の型判断をもつ。これと、$\langle$App$\rangle$ の前提 $\Gamma \vdash_\bullet V : A$ および補題 1 を使うと、期待された結論を導くことができる。

ケース 2 ($\langle$DLet$\rangle$)**.**

サブケース 1 ($\mathsf{dlet}\; p = V : A \;\mathsf{in}\; E[p] \triangleright \mathsf{dlet}\; p = V : A \;\mathsf{in}\; E[V]$)**.** 示したいのは、$\Gamma \vdash_{(\Sigma \setminus \{p\})} \mathsf{dlet}\; p = V : A \;\mathsf{in}\; E[V] : B$ が導出できることである。$\langle$DLet$\rangle$ の前提と補題 2 より、$\Gamma \vdash_\bullet V : A$ および $\Gamma \vdash E[] : A \to B \; \Sigma$（$E[]$ が p を参照する場合）または $\Gamma \vdash E[] : A \to B \; (\Sigma \setminus \{p\})$（$E[]$ が p を参照しない場合）が成立している。これらと補題 3 を使うと、$\Gamma \vdash_\Sigma E[V] : B$ または $\Gamma \vdash_{(\Sigma \setminus \{p\})} E[V] : B$ が導出できる。これに $\langle$DLet$\rangle$ を適用すると、期待された結論が導かれる。

サブケース 2 ($\mathsf{dlet}\; p = V_0 : A \;\mathsf{in}\; V_1 \triangleright V_1$)**.** 示したいのは、$\Gamma \vdash_\bullet V_1 : B$ が成り立つことである。値はエフェクトをもたないため、元の項は $\Gamma \vdash_\bullet \mathsf{dlet}\; p = V_0 : A \;\mathsf{in}\; V_1 : B$、本体 V_1 は $\Gamma \vdash_\bullet V_1 : B$ という型判断をもつはずである。よって、示したい性質は成り立つ。

1 ステップ簡約の型保存が示されたら、簡約ステップ数に関する帰納法によって、もとの定理を示すことができる。 □

次に、進行定理を示す。この性質を示す際には、閉じた値の型からその値の形を導き出す標準形補題（canonical forms lemma）が必要となる。

補題 4 (標準形)**.** $\bullet \vdash_\bullet V : A$ ならば、以下が成り立つ。

1. $A = C$ ならば、$V = c$。
2. $A = A' \to B \; \Sigma$ ならば、$V = \lambda x : A'.\, M$。

定理 2 (進行)**.** $\bullet \vdash_\Sigma M : A$ ならば、M は値であるか、$E[R]$ という形で表されるか、あるいは $E[p]$ という形の stuck した項である。

Proof. M の導出に関する帰納法による。

ケース 1 ($\langle$Const$\rangle$, $\langle$Abs$\rangle$)**.** これらのケースでは主部 M が値であるため、示したい性質は自明に成り立つ。

ケース 2 (⟨Var⟩)**.** このケースは空でない Γ を要求するため、考慮しなくてよい。

ケース 3 (⟨DVar⟩)**.** このケースでは、M が stuck した項であるため、示したい性質は自明に成り立つ。

ケース 4 (⟨App⟩)**.** まず、関数 M が値でない場合を考える。帰納法の仮定より、M は $\mathsf{E}[\mathsf{R}]$ と分解できるか、stuck した項である。前者の場合、関数適用全体は $\mathsf{E}[\mathsf{R}]\ \mathsf{N}$ と表される。後者の場合は、関数適用全体も stuck した項となる。次に、M が値である場合を考える。補題 4 より $\lambda\, \mathsf{x} : \mathsf{A}.\, \mathsf{M}'$ という形をもつ。ここで、引数 N が計算なら、帰納法の仮定より、N は $\mathsf{E}[\mathsf{R}]$ と分解できるか、stuck した項である。前者の場合、関数適用全体は $(\lambda\, \mathsf{x} : \mathsf{A}.\, \mathsf{M}')\ \mathsf{E}[\mathsf{R}]$ と表される。後者の場合は、関数適用全体も stuck した項となる。引数が値であれば、関数適用全体は簡約基とみなされる。

ケース 5 (⟨DLet⟩)**.** まず、本体 M が値でない場合を考える。帰納法の仮定より、M は $\mathsf{E}[\mathsf{R}]$ と分解できるか、stuck した項である。前者の場合、dlet 文全体は $\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ \mathsf{E}[\mathsf{R}]$ と表される。後者の場合は、もし、本体が $\mathsf{E}[\mathsf{p}]$ という形であれば、dlet 文全体は簡約基であり、そうでなければ、dlet 文全体が stuck した項となる。次に、本体が値であれば、項全体は簡約基とみなされる。

□

3 DB の EPS 変換

動的変数は、環境渡し形式（Environment-Passing Style; EPS）への変換によって除去することができる。名前から推測されるように、EPS 変換は、プログラムを「環境を受け取ったら、値を返す」という形の関数に書き換える。こうすることで、動的変数の出現を環境への参照に置き換えることができる。これは、プログラムを「継続を受け取ったら、値を返す」という継続渡し形式（Continuation-Passing Style; CPS）に書き換えることによって、制御演算子を除去できるのと同じ仕組みである。

本節では、DB の選択的な EPS 変換を定義する。ここでいう「選択的な変換」とは、すべての項を EPS に書き換えるのではなく、動的変数への参照を伴う項のみを EPS に書き換えるような変換を指す。制御演算子をもつ計算体系では、実行速度を保ちながら演算子を除去するための方法として、さまざまな選択的 CPS 変換が提案されてきた [28, 31, 2]。これらの変換はいずれも、受け取った項がエフェクトをもつか否かを判断し、その項を CPS あるいは直接形式に書き換える。選択的な EPS 変換はこれまで考えられていないが、DB では動的変数への参照をエフェクトと捉え、その情報を項の導出にもたせているため、選択的 CPS 変換と類似した方法で、動的変数を除去することができる。

DB の EPS 変換は、プログラムの意味と型を保存する。このような変換の存在は、ある言語を動的変数で拡張した際に、既存の評価器が再利用できることを意味する。

3.1 EPS 変換後の言語

本節で定義する EPS 変換は、レコードをもつ単純型付き λ 計算 $\lambda_{\langle\rangle}$ のプログラムを出力する。図 5 に $\lambda_{\langle\rangle}$ の構文と評価規則を示した。$\lambda_{\langle\rangle}$ では、すべての変数が静的なスコープをもち、レコードによって環境が表現される。本稿では、n 個の要素をもつレコード $\langle\langle\langle\rangle, \mathbf{p_1} = \mathbf{A_1}\rangle, ..., \mathbf{p_n} = \mathbf{A_n}\rangle$ を $\langle \mathbf{p_1} = \mathbf{V_1}, ..., \mathbf{p_n} = \mathbf{V_n}\rangle$、その型 $\langle\langle\langle\rangle, \mathbf{p_1} : \mathbf{A_1}\rangle, ..., \mathbf{p_n} : \mathbf{A_n}\rangle$ を $\langle \mathbf{p_1} : \mathbf{A_1}, ..., \mathbf{p_n} : \mathbf{A_n}\rangle$ と表す（ただし、レコードのフィールド名は重複がないものとする）。レコードの値は、$\mathbf{lookup\ p\ P}$ によって取り出される。なお、値環境の定義における $\mathbf{e}$ はレコードを表す変数であり、値の 4 つ目のパターン $\lambda\, \mathbf{e}.\, \mathbf{M}$ はレコードを受け取る関数を表すが、$\mathbf{e}$ に型注釈を付けていないのは、項の同値性判断を容易にするためである。

構文

$$
\begin{array}{lrcl}
\text{項} & \mathbf{M} & ::= & \mathbf{V} \mid \mathbf{M}\ \mathbf{N} \mid \mathbf{M}\ \mathbf{P} \mid \mathbf{lookup}\ \mathbf{p}\ \mathbf{P} \\
\text{値} & \mathbf{V} & ::= & \mathbf{c} \mid \mathbf{x} \mid \lambda\,\mathbf{x} : \mathbf{A}.\,\mathbf{M} \mid \lambda\,\mathbf{e}.\,\mathbf{M} \\
\text{値環境} & \mathbf{P} & ::= & \mathbf{e} \mid \langle\rangle \mid \langle \mathbf{P}, \mathbf{p} = \mathbf{V} \rangle \\
\text{型} & \mathbf{A} & ::= & \mathbf{C} \mid \mathbf{A} \to \mathbf{B} \mid \mathbf{D} \to \mathbf{A} \\
\text{値環境の型} & \mathbf{D} & ::= & \langle\rangle \mid \langle \mathbf{D}, \mathbf{p} : \mathbf{A} \rangle \\
\text{型環境} & \mathbf{\Gamma} & ::= & \bullet \mid \mathbf{\Gamma}, \mathbf{x} : \mathbf{A}
\end{array}
$$

評価文脈

$$
\mathbf{E}[\,] \quad ::= \quad [\,] \mid \mathbf{E}[[\,]\ \mathbf{M}] \mid \mathbf{E}[\mathbf{V}\ [\,]]
$$

簡約規則

$$
\begin{array}{lcll}
(\lambda\,\mathbf{x} : \mathbf{A}.\,\mathbf{M})\ \mathbf{V} & \triangleright & \mathbf{M}[\mathbf{V}/\mathbf{x}] & \\
(\lambda\,\mathbf{e}.\,\mathbf{M})\ \mathbf{P} & \triangleright & \mathbf{M}[\mathbf{P}/\mathbf{e}] & \\
(\lambda\,\mathbf{x} : \mathbf{A}.\,\mathbf{E}[\mathbf{x}])\ \mathbf{M} & \triangleright & \mathbf{E}[\mathbf{M}] & \text{if } \mathbf{x} \notin \mathrm{FV}(\mathbf{E}) \\
\lambda\,\mathbf{e}.\,\mathbf{V}\ \mathbf{e} & \triangleright & \mathbf{V} & \text{if } \mathbf{e} \notin \mathrm{FV}(\mathbf{V}) \\
\mathbf{lookup}\ \mathbf{p}\ \langle \mathbf{P}, \mathbf{p} = \mathbf{V} \rangle & \triangleright & \mathbf{V} & \\
\mathbf{lookup}\ \mathbf{q}\ \langle \mathbf{P}, \mathbf{p} = \mathbf{V} \rangle & \triangleright & \mathbf{lookup}\ \mathbf{q}\ \mathbf{P} & \text{if } \mathbf{p} \neq \mathbf{q}
\end{array}
$$

図 5. $\lambda_{\langle\rangle}$ の構文と評価規則

$\lambda_{\langle\rangle}$ のプログラムは、DB と同様に、値呼び戦略で左から右に実行される。変換の前後において同一の評価戦略を用いているのは、本稿で与える選択的な EPS 変換が評価順序を規定しないことによる。DB は動的束縛以外のエフェクトを含まないが、一般に、プログラムの動作を保存するためには、変換されたプログラムの各部分項が変換前と同じ順序で評価されることを保証しなければならない。

簡約規則を見ると、まず、関数適用に対する規則が 3 つに増えている。このうち、最初の 2 つは β 規則であり、それぞれ通常の λ 抽象と、環境を受け取る関数に適用される。3 つ目は β_Ω 規則とよばれ [32, 21]、λ 抽象の本体において、束縛変数 $\mathbf{x}$ が直ちに実行される位置にあった場合、引数 $\mathbf{M}$ を値に簡約することなく $\mathbf{x}$ に代入できることを表す。これは、正当性の証明において、項と文脈の変換結果を合成する際に使われる。次の η 簡約も、正当性の証明に必要となる規則である。最後の 2 つは、**lookup** の意味論を与えている。

$\lambda_{\langle\rangle}$ はエフェクトをもたないため、型規則は通常の型判断 $\Gamma \vdash \mathsf{M} : \mathsf{A}$ を用いて定義される。図 6 の規則を見ると、レコードに関する規則のほか、部分型付け規則 [Sub] が追加されている。$\lambda_{\langle\rangle}$ では、既存研究 [3, 37] に従って、レコード型 $\mathbf{D_1}$ が $\mathbf{D_2}$ と同じかより多くのフィールドをもち、かつ、2 つの型に共通するフィールドの型が等しい場合、$\mathbf{D_1}$ が $\mathbf{D_2}$ の部分型であるとみなす。この部分型付け関係は、EPS 変換後の項に型を付ける際に使われる。

3.2 選択的 EPS 変換

では、DB から $\lambda_{\langle\rangle}$ への選択的 EPS 変換を定義しよう。冒頭で述べたように、選択的な EPS 変換は、動的変数への参照を伴う項のみを EPS に書き換える。DB では、エフェクトの情報を型判

型規則

$$\frac{\mathbf{c} : \mathbf{C} \in \mathbf{\Phi}}{\mathbf{\Gamma} \vdash \mathbf{c} : \mathbf{C}}\ [\textsc{Const}] \qquad \frac{\mathbf{x} : \mathbf{A} \in \mathbf{\Gamma}}{\mathbf{\Gamma} \vdash \mathbf{x} : \mathbf{A}}\ [\textsc{Var}] \qquad \frac{\mathbf{e} : \mathbf{D} \in \mathbf{\Gamma}}{\mathbf{\Gamma} \vdash \mathbf{e} : \mathbf{D}}\ [\textsc{EVar}]$$

$$\frac{\mathbf{\Gamma}, \mathbf{x} : \mathbf{A} \vdash \mathbf{M} : \mathbf{B}}{\mathbf{\Gamma} \vdash \lambda \mathbf{x} : \mathbf{A}.\, \mathbf{M} : \mathbf{A} \to \mathbf{B}}\ [\textsc{Abs}] \qquad \frac{\mathbf{\Gamma} \vdash \mathbf{M} : \mathbf{A} \to \mathbf{B} \quad \mathbf{\Gamma} \vdash \mathbf{N} : \mathbf{A}}{\mathbf{\Gamma} \vdash \mathbf{M}\ \mathbf{N} : \mathbf{B}}\ [\textsc{App}]$$

$$\frac{\mathbf{\Gamma}, \mathbf{e} : \mathbf{D} \vdash \mathbf{M} : \mathbf{B}}{\mathbf{\Gamma} \vdash \lambda \mathbf{e}.\, \mathbf{M} : \mathbf{D} \to \mathbf{B}}\ [\textsc{EAbs}] \qquad \frac{\mathbf{\Gamma} \vdash \mathbf{M} : \mathbf{D} \to \mathbf{B} \quad \mathbf{\Gamma} \vdash \mathbf{P} : \mathbf{D}}{\mathbf{\Gamma} \vdash \mathbf{M}\ \mathbf{P} : \mathbf{B}}\ [\textsc{EApp}]$$

$$\frac{\mathbf{\Gamma} \vdash \mathbf{V} : \mathbf{A}}{\mathbf{\Gamma} \vdash \langle\rangle : \langle\rangle}\ [\textsc{Empty}] \qquad \frac{\Gamma \vdash \mathbf{P} : \mathbf{D} \quad \mathbf{\Gamma} \vdash \mathbf{V} : \mathbf{A} \quad \mathbf{p} \notin \mathbf{P}}{\mathbf{\Gamma} \vdash \langle \mathbf{P}, \mathbf{p} = \mathbf{V} \rangle : \langle \mathbf{D}, \mathbf{p} : \mathbf{A} \rangle}\ [\textsc{Extend}]$$

$$\frac{\mathbf{\Gamma} \vdash \mathbf{P} : \langle \mathbf{p_1} : \mathbf{A_1}, \ldots, \mathbf{p_n} : \mathbf{A_n} \rangle}{\mathbf{\Gamma} \vdash \mathbf{lookup}\ \mathbf{p_i}\ \mathbf{P} : \mathbf{A_i}}\ [\textsc{Lookup}] \qquad \frac{\mathbf{\Gamma} \vdash \mathbf{P} : \mathbf{D} \quad \mathbf{D} \leq \mathbf{D'}}{\mathbf{\Gamma} \vdash \mathbf{P} : \mathbf{D'}}\ [\textsc{Sub}]$$

部分型規則

$$\frac{}{\mathbf{D} \leq \mathbf{D}}\ [\textsc{Refl}] \qquad \frac{\mathbf{D_0} \leq \mathbf{D_1} \quad \mathbf{D_1} \leq \mathbf{D_2}}{\mathbf{D_0} \leq \mathbf{D_2}}\ [\textsc{Trans}] \qquad \frac{\forall \mathbf{p} : \mathbf{A} \in \mathbf{D'}.\, \mathbf{p} : \mathbf{A} \in \mathbf{D}}{\mathbf{D} \leq \mathbf{D'}}\ [\textsc{Env}]$$

図 6. $\lambda_{\langle\rangle}$ の型規則

断にもたせているため、受け取った項を EPS に変換するべきかどうかは、その項の導出を見れば判断できる。つまり、M が $\Gamma \vdash_\bullet M : A$ という導出をもつときは直接形式の項を、$\Gamma \vdash_\Sigma M : A$（ただし、$\Sigma$ は空でない環境）という導出をもつときは環境を受け取る関数 $\lambda \mathbf{e}.\, \mathbf{M}$ を返すようにすれば、選択的な変換が得られる。本稿では見やすさのため、以下の表記を用いる。

$$M^+ \stackrel{\text{def}}{\equiv} \mathbf{M} \text{ such that } \Gamma \vdash_\bullet M : A \overset{+}{\rightsquigarrow} \mathbf{M}$$
$$M^\div \stackrel{\text{def}}{\equiv} \mathbf{M} \text{ such that } \Gamma \vdash_\Sigma M : A \overset{\div}{\rightsquigarrow} \mathbf{M}$$

図 7 - 8 に項と型の変換規則を示した。ここで、$+$ は直接形式への変換、$\div$ は EPS への変換を表す。定数と静的変数は値であるため、これらは直接形式の項に変換される。一方、動的変数は「環境 $\mathbf{e}$ を受け取ったら、その中から $\mathbf{p}$ の値を探し出す」という関数に書き換えられる。関数は、本体がエフェクトをもつかどうかによって、異なる形に変換される。本体 M が純粋であった場合、M には直接形式の変換が適用されるため、変換後の関数は引数 $\mathbf{x}$ のみを受け取る。一方、本体が純粋でない場合、M は EPS に変換されるため、変換後の関数は引数を受け取った後、さらに本体の計算に使う環境を要求する。

関数適用は、関数、引数、関数本体のエフェクトの組み合わせによって、8 通りの変換をもつ。このうち、すべての部分項が純粋である場合、関数適用全体もエフェクトを伴わないため、変換結果は直接形式の関数適用となる。残りのケースのうち、はじめの 3 つは関数本体が純粋である場合に対応し、受け取った環境 $\mathbf{e}$ が関数部分と引数部分でのみ使われる。一方、後半の 4 つは、関数本体がエフェクトをもつ場合であり、環境 $\mathbf{e}$ が関数適用の結果に渡されている。

dlet 文には 4 種類の変換規則が与えられている。まず、本体 M が純粋である場合は、$p = V$ と

$$
\begin{array}{rcll}
\langle \textsc{Const} \rangle & \overset{+}{\rightsquigarrow} & \mathbf{c} & \\
\langle \textsc{Var} \rangle & \overset{+}{\rightsquigarrow} & \mathbf{x} & \\
\langle \textsc{DVar} \rangle & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, \mathbf{lookup}\ \mathbf{p}\ \mathbf{e} & \\
\langle \textsc{Abs} \rangle & \overset{+}{\rightsquigarrow} & \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \mathsf{M}^{+} & \text{if } \Sigma \text{ empty} \\
 & \overset{+}{\rightsquigarrow} & \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \mathsf{M}^{\div} & \text{if } \Sigma \text{ non-empty} \\
\langle \textsc{App} \rangle & \overset{+}{\rightsquigarrow} & \mathsf{M}^{+}\ \mathsf{N}^{+} & \text{if } \Sigma_0, \Sigma_1, \Sigma_2 \text{ empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, \mathsf{M}^{+}\ (\mathsf{N}^{\div}\ \mathbf{e}) & \text{if } \Sigma_0, \Sigma_2 \text{ empty, } \Sigma_1 \text{ non-empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, (\mathsf{M}^{\div}\ \mathbf{e})\ \mathsf{N}^{+} & \text{if } \Sigma_1, \Sigma_2 \text{ empty, } \Sigma_0 \text{ non-empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, (\mathsf{M}^{\div}\ \mathbf{e})\ (\mathsf{N}^{\div}\ \mathbf{e}) & \text{if } \Sigma_2 \text{ empty, } \Sigma_0, \Sigma_1 \text{ non-empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, \mathsf{M}^{+}\ \mathsf{N}^{+}\ \mathbf{e} & \text{if } \Sigma_0, \Sigma_1 \text{ empty, } \Sigma_2 \text{ non-empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, \mathsf{M}^{+}\ (\mathsf{N}^{\div}\ \mathbf{e})\ \mathbf{e} & \text{if } \Sigma_0 \text{ empty, } \Sigma_1, \Sigma_2 \text{ non-empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, (\mathsf{M}^{\div}\ \mathbf{e})\ \mathsf{N}^{+}\ \mathbf{e} & \text{if } \Sigma_1 \text{ empty, } \Sigma_0, \Sigma_2 \text{ non-empty} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, (\mathsf{M}^{\div}\ \mathbf{e})\ (\mathsf{N}^{\div}\ \mathbf{e})\ \mathbf{e} & \text{if } \Sigma_0, \Sigma_1, \Sigma_2 \text{ non-empty} \\
\langle \textsc{DLet} \rangle & \overset{+}{\rightsquigarrow} & \mathsf{M}^{+} & \text{if } \Sigma \text{ empty} \\
 & \overset{+}{\rightsquigarrow} & \mathsf{M}^{\div}\ \langle \mathbf{p} = \mathsf{V}^{+} \rangle & \text{if } \Sigma = \bullet,\ \mathsf{p} : \mathsf{A} \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, \mathsf{M}^{\div}\ \mathbf{e} & \text{if } \Sigma \text{ non-empty and } \mathsf{p} \notin \Sigma \\
 & \overset{\div}{\rightsquigarrow} & \lambda\, \mathbf{e}.\, \mathsf{M}^{\div}\ \langle \mathbf{e}, \mathbf{p} = \mathsf{V}^{+} \rangle & \text{if } \Sigma \setminus \{\mathsf{p}\} \text{ non-empty}
\end{array}
$$

図 7. 項の EPS 変換

いう束縛を無視し、M に直接形式への変換を適用する。M が p のみを参照するような項であった場合は、M を EPS に変換したうえで、$\mathbf{p} = \mathsf{V}^{+}$ という情報をもった環境を渡す。残りのケースは、M が p 以外の動的変数を参照する場合に対応する。両者とも、出力が環境 $\mathbf{e}$ を受け取る関数となっているが、M が p にアクセスするかどうかによって、$\mathsf{M}^{\div}$ に異なる環境（$\mathbf{e}$ または $\mathbf{e}, \mathbf{p} : \mathsf{V}^{+}$）が渡されている。

次に、型の変換を見てみよう。DB の型はエフェクトをもたないため、表層構造を見れば、どのように EPS 変換するべきかが分かる。そのため、型の変換は構文に関して帰納的に定義される。具体的な規則を見ると、定数型 C は、対応する $\lambda_{\langle\rangle}$ の定数型 $\mathbf{C}$ に置き換えられる。関数型は、本体のエフェクトを表す Σ が空であった場合は $\mathsf{A}^{+} \to \mathsf{B}^{+}$ に、空でない場合は $\mathsf{A}^{+} \to \Sigma^{+} \to \mathsf{B}$ に変換される。この 2 つの規則は、λ 抽象の 2 つの変換と対応しており、変換後の関数本体を計算する際に環境が必要となるかどうかを表している。なお、Σ に対する変換は、関数本体がエフェクトを伴うケースのみで使われるため、空の場合を定義する必要はない。

変数の型注釈と EPS 変換　2.2 節でふれたように、DB の束縛変数に型注釈を付けているのは、1 つの項が EPS 変換によって異なる項に書き換えられるのを防ぐためである。例として、$\lambda\, \mathsf{f}.\, \lambda\, \mathsf{g}.\, (\mathsf{f}\ 0) + (\mathsf{f}\ 1)$ という項を考えよう（ただし、DB が足し算をサポートすることを仮定している）。この項の本体は、任意の Σ_0, Σ_1 に対して、以下のような型判断をもつ：

$$
\begin{aligned}
\mathsf{C} &\overset{+}{\rightsquigarrow} \mathbf{C} && \\
\mathsf{A} \to \mathsf{B}\ \Sigma &\overset{+}{\rightsquigarrow} \mathsf{A}^+ \to \mathsf{B}^+ && \text{if } \Sigma \text{ empty} \\
&\overset{+}{\rightsquigarrow} \mathsf{A}^+ \to \Sigma^+ \to \mathsf{B}^+ && \text{if } \Sigma \text{ non-empty} \\
\\
\bullet, \mathsf{p} : \mathsf{A} &\overset{+}{\rightsquigarrow} \langle \mathbf{p} : \mathsf{A}^+ \rangle && \\
\Sigma, \mathsf{p} : \mathsf{A} &\overset{+}{\rightsquigarrow} \langle \Sigma^+, \mathbf{p} : \mathsf{A}^+ \rangle &&
\end{aligned}
$$

図 8. 型の EPS 変換

$$
\bullet, \mathsf{f} : \mathsf{int} \to \mathsf{int}\ \Sigma_0, \mathsf{g} : \mathsf{int} \to \mathsf{int}\ \Sigma_1 \vdash_{(\Sigma_0 \cup \Sigma_1)} (\mathsf{f}\ 0) + (\mathsf{g}\ 1) : \mathsf{int}
$$

ここで、足し算の EPS 変換は、関数適用の本体が純粋であるケースと同じような規則によって定義される。よって、$(\mathsf{f}\ 0) + (\mathsf{g}\ 1)$ は、Σ_0 と Σ_1 がそれぞれ空であるか、そうでないかによって、4 通りの異なる変換結果をもつ。すると、変換の正当性を議論する際に、図 5 の簡約規則より強い同値性の概念が必要となる。

一方、f と g に型注釈をもたせた場合、これらが純粋であるか否かが明確になるため、EPS 変換の結果は一意に定まる。

3.3 EPS 変換の性質

3.2 節で定義した EPS 変換は、プログラムの意味を保存する。これを示すためには、2 つの補題が必要となる。なお、2 つ目の補題で使われる評価文脈の変換は、付録に掲載している。

補題 5 (変換と代入)**.** EPS 変換と代入は可換である。つまり、以下が成り立つ。

1. $(\mathsf{M}[\mathsf{V}/\mathsf{x}])^+ = \mathsf{M}^+[\mathsf{V}^+/\mathbf{x}]$
2. $(\mathsf{M}[\mathsf{V}/\mathsf{x}])^\div = \mathsf{M}^\div[\mathsf{V}^+/\mathbf{x}]$

ただし、$=$ は図 5 の簡約規則によって定義される同値性である（以降も同様）。

補題 6 (合成された項の変換)**.** $\Gamma \vdash_{\Sigma_0} \mathsf{M} : \mathsf{A}$ と $\Gamma \vdash \mathsf{E}[\,] : \mathsf{A} \to \mathsf{B}\ \Sigma_1$ が導出可能であるとする。

1. $\Sigma_0, \Sigma_1 = \bullet$ ならば、$(\mathsf{E}[\mathsf{M}])^+ = \mathsf{E}[\,]^+\ \mathsf{M}^+$。
2. $\Sigma_0 \neq \bullet, \Sigma_1 = \bullet$ かつ $\Gamma \vdash_\bullet \mathsf{E}[\mathsf{M}] : \mathsf{B}$ ならば、$(\mathsf{E}[\mathsf{M}])^+ = \mathsf{E}[\,]^+\ (\mathsf{M}^\div\ (\text{build-env}(\mathsf{E}[\,], \Sigma_0)))$。
3. $\Sigma_0 \neq \bullet, \Sigma_1 = \bullet$ かつ $\Gamma \vdash_\Sigma \mathsf{E}[\mathsf{M}] : \mathsf{B}$ ならば、$(\mathsf{E}[\mathsf{M}])^\div = \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^+\ (\mathsf{M}^\div\ \langle \mathbf{e}, \text{build-env}(\mathsf{E}[\,], \Sigma_0) \rangle)$
4. $\Sigma_0 = \bullet, \Sigma_1 \neq \bullet$ ならば、$(\mathsf{E}[\mathsf{M}])^\div = \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^\div\ \mathsf{M}^+\ \mathbf{e}$。
5. $\Sigma_0, \Sigma_1 \neq \bullet$ ならば、$(\mathsf{E}[\mathsf{M}])^\div = \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^\div\ (\mathsf{M}^\div\ \langle \mathbf{e}, \text{build-env}(\mathsf{E}[\,], \Sigma_0) \rangle)\ \mathbf{e}$。

ただし、$\text{build-env}(\mathsf{E}[\,], \Sigma)$ は以下のように定義される。

$$
\begin{aligned}
\text{build-env}([\,], \Sigma) &= \langle\,\rangle \\
\text{build-env}(\mathsf{E}[[\,]\ \mathsf{N}], \Sigma) &= \text{build-env}(\mathsf{E}[\,], \Sigma) \\
\text{build-env}(\mathsf{E}[\mathsf{V}\ [\,]], \Sigma) &= \text{build-env}(\mathsf{E}[\,], \Sigma) \\
\text{build-env}(\mathsf{E}[\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ [\,]], \Sigma) &= \langle \text{build-env}(\mathsf{E}[\,], (\Sigma \setminus \{\mathsf{p}\})), \mathbf{p} = \mathsf{V}^+ \rangle && \text{if } \mathsf{p} \in \Sigma \\
&= \text{build-env}(\mathsf{E}[\,], \Sigma) && \text{if } \mathsf{p} \notin \Sigma
\end{aligned}
$$

定理 3 (正当性)**.**

1. $\Gamma \vdash_\bullet M : A$ かつ $M \triangleright^* M'$ ならば、$M^+ = M'^+$。
2. $\Gamma \vdash_\Sigma M : A$ かつ $M \triangleright^* M'$ ならば、$M^\div = M'^\div$。

Proof. DB の型保存定理と同じように、1 ステップの簡約に関する正当性を示したのち、その結果を任意の簡約ステップに拡張する。ここでは、$\mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[p] \triangleright \mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[V]$ のケースのうち、dlet 文全体が純粋である場合を示す。

ケース 1 ($\Gamma \vdash E[] : B \to A'\ \bullet$)**.**

$$
\begin{aligned}
&(\mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[p])^+ \\
&= (E[p])^\div\ \langle \mathbf{p} = V^+ \rangle && \text{変換の定義} \\
&= (\lambda \mathbf{e}.\, E[]^+\ ((\lambda \mathbf{e}.\, \mathbf{lookup}\ \mathbf{p}\ \mathbf{e})\ \mathbf{e}))\ \langle \mathbf{p} = V^+ \rangle && \text{変換の定義} \\
&\triangleright E[]^+\ (\mathbf{lookup}\ \mathbf{p}\ \langle \mathbf{p} = V^+ \rangle) && \beta \text{ 簡約} \\
&\triangleright E[]^+\ V^+ && \text{lookup の簡約} \\
&= (E[V])^+ && \text{補題 6} \\
&= (\mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[V])^+ && \text{変換の定義}
\end{aligned}
$$

ケース 2 ($\Gamma \vdash E[] : B \to A'\ (\bullet, p : A)$)**.**

$$
\begin{aligned}
&(\mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[p])^+ \\
&= (E[p])^\div\ \langle \mathbf{p} = V^+ \rangle && \text{変換の定義} \\
&= (\lambda \mathbf{e}.\, E[]^\div\ ((\lambda \mathbf{e}.\, \mathbf{lookup}\ \mathbf{p}\ \mathbf{e})\ \mathbf{e})\ \mathbf{e})\ \langle \mathbf{p} = V^+ \rangle && \text{変換の定義} \\
&\triangleright E[]^\div\ (\mathbf{lookup}\ \mathbf{p}\ \langle \mathbf{p} = V^+ \rangle)\ \langle \mathbf{p} = V^+ \rangle && \beta \text{ 簡約} \\
&\triangleright E[]^\div\ V^+\ \langle \mathbf{p} = V^+ \rangle && \text{lookup の簡約} \\
&\triangleleft (\lambda \mathbf{e}.\, E[]^\div\ V^+\ \mathbf{e})\ \langle \mathbf{p} = V^+ \rangle && \beta \text{ 簡約} \\
&= (E[V])^\div\ \langle \mathbf{p} = V^+ \rangle && \text{補題 6} \\
&= (\mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[V])^+ && \text{変換の定義}
\end{aligned}
$$

□

また、型付け可能な DB のプログラムは、型付け可能な $\lambda_{\langle\rangle}$ のプログラムに変換される。

定理 4 (型保存)**.**

1. $\Gamma \vdash_\bullet M : A \Rightarrow \Gamma^+ \vdash M^+ : A^+$
2. $\Gamma \vdash_\Sigma M : A \Rightarrow \Gamma^+ \vdash M^\div : \Sigma^+ \to A^+$

Proof. 導出の長さに関する帰納法による。唯一注意しなければならないのは、関数適用のケースのうち、エフェクトをもつ項が複数存在する場合である。例として、関数 M と引数 N がともにエフェクトをもち、関数本体がエフェクトをもたないケース（4 番目の変換規則）を考えよう。DB の型付け規則 (App) より、関数適用全体の実行には、部分項の実行に必要な環境の和が用いられる。したがって、M と N の導出に用いられる環境が Σ_0 と Σ_1 だった場合、変換結果 $\lambda \mathbf{e}.\, (M^\div\ \mathbf{e})\ (N^\div\ \mathbf{e})$ に現れる $\mathbf{e}$ は $(\Sigma_0 \cup \Sigma_1)^+$ 型をもつ。この環境は、部分項の EPS 変換結果に分配されるが、帰納法の仮定より、$M^\div$ と $N^\div$ はこれより小さい ${\Sigma_0}^+$, ${\Sigma_1}^+$ 型の環境を要求する。ここで、$\lambda_{\langle\rangle}$ に部分型の規則 [Sub] をもたせたのを思い出そう。[Env] より、$\mathbf{e}$ の型は ${\Sigma_0}^+$, ${\Sigma_1}^+$ の部分型である。これは、[Sub] を使って、$\mathbf{e}$ の型を $M^\div$, $N^\div$ が要求する型に書き換えられることを意味する。したがって、$(M\ N)^\div$ には正しく型が付く。 □

4　依存型付き言語：DDB

2 節と 3 節では、単純型付き λ 計算の中で、動的変数をどのようにエフェクトとして扱うか、また、これらをどのように除去できるかを見てきた。本節では、DB を依存型で拡張した言語 DDB を定義する。1 節で述べたように、依存型付き言語の中でエフェクトを扱う場合、型の依存性に関する制約が必要となる。本研究では、型の中に現れる項を、動的変数への参照を伴わないものに制限する。こうすることで、型の意味が静的に定まることを保証する。

4.1　依存型と動的変数の相互作用

まず、依存型をもつ計算体系で動的変数を無制限に使用するとどのような問題が起きるかを見てみよう。依存型付き λ 計算において、関数適用 $M\ N$ は次のような型規則をもつ：

$$\frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B[N/x]}\ (\textsc{DApp})$$

関数 M の型 $\Pi x : A.\, B$ は $A \to B$ を一般化したものであり、引数を表す変数 x が関数の返す型 B に出現することが許される。この x は、結論部において実際の引数 N に置き換わる。

ここで、自然数 n を受け取ったら、長さが n のリストを返す関数 f が定義されているとする。今、自然数の型を $\mathbb{N}$、長さが n のリストの型を $L\ n$ と表すと、f の型は $\Pi n : \mathbb{N}.\, L\ n$ となる。この関数を $\mathbb{N}$ 型の動的変数 p に適用したとしよう。規則 (DApp) より、$f\ p$ は $L\ p$ という型をもつ。この型はどのように解釈されるだろうか。項のレベルにおいて、動的変数 p の値が定まるのは、関数適用 $f\ p$ を含むプログラムが $\mathsf{dlet}\ p = V\ \mathsf{in}\ E[f\ p]$ という形に実行されたときである。しかし、$f\ p$ を取り囲む文脈の情報は、この項を型付けする際には得られないため、$L\ p$ に含まれる p は特定の値に置き換えられない。よって、$L\ p$ は「長さが不明のリストの型」と解釈される。

$f\ p$ のような、意味が完全に定まっていない型をもつ項は、プログラム全体の型付け可能性を静的に定めることの妨げとなる。たとえば、空でないリストを受け取ったら、その先頭要素を返す関数 g が定義されていると仮定しよう。あるリストが空でないということは、その型のインデックス（$L\ n$ の n）が 1 以上であることに対応する。これをふまえて、関数適用 $g\ (f\ p)$ を考えると、この項は $\mathsf{dlet}\ p = 1\ \mathsf{in}\ []$ という文脈に囲まれたときは型が付くが、$\mathsf{dlet}\ p = 0\ \mathsf{in}\ []$ に囲まれたときは型が付かない。依存型付き言語では、プログラムの信頼性を高める目的で型が使われるため、型付け可能性が文脈という動的な要素に依存することは望ましくない。したがって、型に現れる項は、文脈によらず値が定まる純粋な項に制限するべきである。本節で定義する DDB では、型規則を用いて、動的変数に依存した型の生成を防ぐというアプローチをとる。

4.2　DDB の構文・簡約規則・同値性

DDB は Harper ら [16] の LF に動的変数を加えたものであり、項の構成要素は DB と同様であるが、型の定義が異なるほか、種というカテゴリをもつ。図 9 を見てみよう。まず、DDB では型を返す関数 $\lambda \mathsf{x} : \mathsf{A}.\, \mathsf{B}$ と関数適用 $\mathsf{A}\ \mathsf{M}$ を使って、型レベルの計算を表現できる（ただし、引数は項に限定されている）。$\Pi \mathsf{x} : \mathsf{A}.\, \mathsf{B}$ Σ は依存性を伴う関数型を表し、変数 x が B と Σ 中の型に現れ得る。種の定義の中で、$*$ は型の型、$\Pi \mathsf{x} : \mathsf{A}.\, \mathsf{K}$ は型レベルの関数の型を表す。後者の値域に注釈 Σ が現れないのは、型がエフェクトをもたないことによる。本稿では以降、メタ変数 T を「種、型、あるいは項」という意味で使用する。

DDB の計算は、DB と同様に、値呼び戦略で左から右へ評価される。β 簡約の関数本体が M から T になったのは、関数適用の結果として型が返される場合があることによる。

依存型付き言語では、項に型を付ける際に、同値性に基づいて型を置き換えるという操作がしばしば行われる。DDB では、同値性 $\equiv$ を Takahashi [35] の parallel reduction $\triangleright_p$ によって定義す

構文

$$
\begin{array}{lll}
\text{型環境} & \Gamma & ::= \bullet \mid \Gamma, x : A \\
\text{動的変数の環境} & \Sigma & ::= \bullet \mid \Sigma, p : A \\
\text{種} & K & ::= * \mid \Pi x : A.\, K \\
\text{型} & A & ::= C \mid \Pi x : A.\, B\ \Sigma \mid \lambda x : A.\, B \mid A\ M \\
\text{項} & M & ::= V \mid p \mid M\ N \mid \mathsf{dlet}\ p = V : A\ \mathsf{in}\ M \\
\text{値} & V & ::= c \mid x \mid \lambda x : A.\, M
\end{array}
$$

簡約規則

$$
\begin{array}{lcl}
(\lambda x : A.\, T)\ V & \triangleright & T[V/x] \\
\mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[p] & \triangleright & \mathsf{dlet}\ p = V : A\ \mathsf{in}\ E[V] \quad \text{if } p \notin \mathrm{BP}(E) \\
\mathsf{dlet}\ p = V_0 : A\ \mathsf{in}\ V_1 & \triangleright & V_1
\end{array}
$$

同値性規則

$$
\frac{T_0 \ \triangleright_p^*\ T \qquad T_1 \ \triangleright_p^*\ T}{T_0 \equiv T_1}\ \langle \equiv \rangle
$$

図 9. DDB の構文、簡約規則、同値性規則

る。この簡約関係のもとでは、複数の部分項の簡約を並行して行うことができる（具体的な定義は付録を参照されたい）。図 9 の規則 $\langle \equiv \rangle$ は、「2 つの表現 T_0, T_1 が等しいのは、これらが $\triangleright_p$ によって共通の表現 T に簡約されるときである」ということを意味する。

4.3 DDB の型規則

DDB は依存型をもつため、項に対する型付け規則に加えて、環境や種、型に対する導出規則が必要となる。図 10 にこれらの規則を示す。型環境に関する規則のうち、$\langle$G-Ext$\rangle$ の 2 つ目の前提は、新たに追加する型 A が、拡張前の環境 Γ のもとで導出可能であることを表している。これによって、A から Γ に含まれる変数への参照が可能となる。なお、ここで動的変数の環境が使われていないのは、型が動的変数に依存することを許さないという制限を設けたためである。動的変数の環境 Σ を拡張する際も同様に、追加する型の導出可能性に型環境 Γ のみを用いている。種と型に関する規則の中で、$\langle$K-Pi$\rangle$ と $\langle$A-Pi$\rangle$ は、K, B, Σ の導出に拡張された型環境 $\Gamma, x : A$ を使うことで、値域から引数 x にアクセスすることを可能にしている。関数適用の規則 $\langle$A-App$\rangle$ に注目すると、引数 M のエフェクトが $\bullet$ に固定されている。これは、動的変数に依存する型が導出されるのを防ぐ役割をもつ。また、この関数適用は M に依存した型 $K[M/x]$ をもつが、型規則の中で、この種が Γ のもとで導出できることを検査する必要がある。なぜなら、DDB は値呼びの言語であり、変数に値でない M を代入すると、代入前に成立していた同値関係が壊され、導出不可能な種 $K[M/x]$ が結論部に現れてしまうことがあるからである [1]。最後の規則 $\langle$A-Conv$\rangle$ は、同値性 $K \equiv K'$ を使って型 A の種 K を K' に置き換えることを許す。

[1] たとえば、loop が停止しない計算であったとすると、$(\lambda x : A.\, c)\ y \equiv c$ は成立するが、$(\lambda x : A.\, c)\ y[\mathsf{loop}/y] \equiv c[\mathsf{loop}/y]$ は成立しない。そのため、K の導出が代入前の同値関係に依存していた場合、$\Gamma, x : A \vdash K$ は導出できるが、$\Gamma \vdash K[M/x]$ は導出できない。2.4 節の議論から、DDB で loop のような項を定義するのは不可能であると考えられるが、停止性の証明はまだ行っていないため、ここでは計算の代入を安全とみなしていない。

型環境 $\vdash \Gamma$

$$\frac{}{\vdash \bullet}\ \langle\text{G-Empty}\rangle \qquad \frac{\vdash \Gamma \quad \Gamma \vdash A : *}{\vdash \Gamma, x : A}\ \langle\text{G-Ext}\rangle$$

動的変数の環境 $\Gamma \vdash \Sigma$

$$\frac{\vdash \Gamma}{\Gamma \vdash \bullet}\ \langle\text{S-Empty}\rangle \qquad \frac{\Gamma \vdash \Sigma \quad \Gamma \vdash A : *}{\Gamma \vdash \Sigma, p : A}\ \langle\text{S-Ext}\rangle$$

種 $\Gamma \vdash K$

$$\frac{\vdash \Gamma}{\Gamma \vdash *}\ \langle\text{K-Star}\rangle \qquad \frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash K}{\Gamma \vdash \Pi x : A.\, K}\ \langle\text{K-Pi}\rangle$$

型 $\Gamma \vdash A : K$

$$\frac{\vdash \Gamma \quad C : K \in \Phi}{\Gamma \vdash C : K}\ \langle\text{A-Base}\rangle \qquad \frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : * \quad \Gamma \vdash \Sigma}{\Gamma \vdash \Pi x : A.\, B\ \Sigma : *}\ \langle\text{A-Pi}\rangle$$

$$\frac{\Gamma, x : A \vdash B : K}{\Gamma \vdash \lambda x : A.\, B : (\Pi x : A.\, K)}\ \langle\text{A-Abs}\rangle \qquad \frac{\Gamma \vdash A : (\Pi x : B.\, K) \quad \Gamma \vdash_{\bullet} M : B \quad \Gamma \vdash K[M/x]}{\Gamma \vdash A\ M : K[M/x]}\ \langle\text{A-App}\rangle$$

$$\frac{\Gamma \vdash A : K \quad \Gamma \vdash K' \quad K \equiv K'}{\Gamma \vdash A : K'}\ \langle\text{A-Conv}\rangle$$

図 10. DDB の型付け規則

項の導出規則（図 11）は DB の規則と同じような構造をもつが、依存型に対応するために、いくつか新しい前提が追加されている。まず、ベースとなるケース ⟨M-Const⟩ と ⟨M-Var⟩ において、型環境 Γ の導出可能性を明示的に検査することで、結論部の型環境に自由変数をもつ型や、動的変数に依存した型が現れないことを保証している。次に、関数適用の規則 ⟨M-App⟩ の中で、関数が依存した型を持つ場合に純粋な引数を要求することで、動的変数に依存した型が結論部に現れることを防いでいる。最後に、⟨M-Conv⟩ は、項がもつ型と、そのエフェクトを書き換えるための規則である。ここで、$\Sigma \equiv \Sigma'$ が成り立つのは、両者に含まれる型がすべて同値であるときとする。

4.4 健全性

DDB の健全性は、依存型が加わったことにより、証明が複雑になるものの、2.5 の補題を適宜拡張することで示すことができる。特に、λ 計算のフラグメントに関しては、Sjöberg ら [33] や Casinghino ら [4] の証明と同じ流れとなる。

項 $\Gamma \vdash_\Sigma M : A$

$$\frac{\vdash \Gamma \quad c : C \in \Phi}{\Gamma \vdash_\bullet c : C}\ \langle \text{M-Const} \rangle \qquad \frac{\vdash \Gamma \quad x : A \in \Gamma}{\Gamma \vdash_\bullet x : A}\ \langle \text{M-Var} \rangle \qquad \frac{\Gamma \vdash A : *}{\Gamma \vdash_{(\bullet,\, p:A)} p : A}\ \langle \text{M-DVar} \rangle$$

$$\frac{\Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\bullet \lambda x : A.\, M : (\Pi x : A.\, B\ \Sigma)}\ \langle \text{M-Abs} \rangle$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\Sigma_0} M : (\Pi x : A.\, B\ \Sigma_2) \quad \Gamma \vdash_{\Sigma_1} N : A \\ \Sigma_1 = \bullet \text{ if } x \in \mathrm{FV}(B) \cup \mathrm{FV}(\Sigma_2) \\ \Gamma \vdash B[N/x] : * \quad \Gamma \vdash \Sigma_2[N/x] \quad \mathsf{consistent}\,(\Sigma_0 \cup \Sigma_1 \cup \Sigma_2[N/x])\end{array}}{\Gamma \vdash_{(\Sigma_0 \cup \Sigma_1 \cup \Sigma_2[N/x])} M\ N : B[N/x]}\ \langle \text{M-App} \rangle$$

$$\frac{\Gamma \vdash_\bullet V : A \quad \Gamma \vdash_\Sigma M : B \quad \text{if } p : A' \in \Sigma,\ A' = A}{\Gamma \vdash_{(\Sigma \setminus \{p\})} \mathsf{dlet}\ p = V : A\ \mathsf{in}\ M : B}\ \langle \text{M-DLet} \rangle$$

$$\frac{\begin{array}{c}\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash A' : * \quad \Gamma \vdash \Sigma' \\ A \equiv A' \quad \Sigma \equiv \Sigma'\end{array}}{\Gamma \vdash_{\Sigma'} M : A'}\ \langle \text{M-Conv} \rangle$$

図 11. DDB の型付け規則（続き）

定理 5 (型保存)**.**

1. $\Gamma \vdash K$ かつ $K \rhd_p^* K'$ ならば、$\Gamma \vdash K'$。
2. $\Gamma \vdash A : K$ かつ $A \rhd_p^* A'$ ならば、$\Gamma \vdash A' : K$。
3. $\Gamma \vdash_\Sigma M : A$ かつ $M \rhd_p^* M'$ ならば、$\Gamma \vdash_\Sigma M' : A$。

型保存定理は parallel reduction に関する性質となり、種や型のレベルにも拡張される。この定理を示す際には、まず、代入補題を変更する必要がある。これは、項レベルの代入が型レベルに影響を与えることによる。次に、ある項に A という型が付いたら、A は同じ環境のもとで導出できることを示さなければならない。この性質は、⟨A-Conv⟩ と ⟨M-Conv⟩ によって型を付け替えるときに使われる。

定理 6 (進行)**.** $\bullet \vdash_\Sigma M : A$ ならば、M は値であるか、$E[R]$ という形で表されるか、あるいは $E[p]$ という形の stuck した項である。

進行定理の文言は定理 2 と同じだが、標準形補題を示す際に、同値性が健全であるという性質が必要となる。なぜなら、⟨M-Conv⟩ によって付け替えられた型が元の型と異なる構造をもつと、帰納法の仮定が使えなくなってしまうからである。

5 DDB の EPS 変換

では、3 節と同じ方法で、DDB に対する選択的 EPS 変換を定義しよう。

$$
\begin{array}{lrcl}
\text{型環境} & \Gamma & ::= & \bullet \mid \Gamma, \mathbf{x} : \mathbf{A} \\
\text{種} & \mathbf{K} & ::= & * \mid \Pi\, \mathbf{x} : \mathbf{A}.\, \mathbf{K} \\
\text{型} & \mathbf{A} & ::= & \mathbf{C} \mid \Pi\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \mid \mathbf{D} \to \mathbf{A} \mid \lambda\, \mathbf{x} : \mathbf{A}.\, \mathbf{B} \mid \mathbf{A}\ \mathbf{B} \\
\text{項} & \mathbf{M} & ::= & \mathbf{V} \mid \mathbf{M}\ \mathbf{N} \mid \mathbf{M}\ \mathbf{P} \mid \mathbf{lookup}\ \mathbf{p}\ \mathbf{P} \\
\text{値} & \mathbf{V} & ::= & \mathbf{c} \mid \mathbf{x} \mid \lambda\, \mathbf{x} : \mathbf{A}.\, \mathbf{M} \mid \lambda\, \mathbf{e}.\, \mathbf{M} \\
\text{値環境} & \mathbf{P} & ::= & \mathbf{e} \mid \bullet \mid \langle \mathbf{P}, \mathbf{p} = \mathbf{V} \rangle \\
\text{環境の型} & \mathbf{D} & ::= & \langle\,\rangle \mid \langle \mathbf{D}, \mathbf{p} : \mathbf{A} \rangle
\end{array}
$$

図 12. $d\lambda_{\langle\rangle}$ の構文

$$
\begin{array}{rcll}
\langle \text{K-Star} \rangle & \overset{+}{\rightsquigarrow} & * & \\
\langle \text{K-Pi} \rangle & \overset{+}{\rightsquigarrow} & \Pi\, \mathbf{x} : \mathsf{A}^{+}.\, \mathsf{K}^{+} & \\
\\
\langle \text{A-Pi} \rangle & \overset{+}{\rightsquigarrow} & \Pi\, \mathbf{x} : \mathsf{A}^{+}.\, \mathsf{B}^{+} & \text{if } \Sigma \text{ empty} \\
 & \overset{+}{\rightsquigarrow} & \Pi\, \mathbf{x} : \mathsf{A}^{+}.\, \Sigma^{+} \to \mathsf{B}^{+} & \text{if } \Sigma \text{ non-empty} \\
\langle \text{A-Abs} \rangle & \overset{+}{\rightsquigarrow} & \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \mathsf{B}^{+} & \\
\langle \text{A-App} \rangle & \overset{+}{\rightsquigarrow} & \mathsf{A}^{+}\ \mathsf{M}^{+} & \\
\langle \text{A-Conv} \rangle & \overset{+}{\rightsquigarrow} & \mathsf{A}^{+} & \\
\\
\langle \text{M-Conv} \rangle & \overset{+}{\rightsquigarrow} & \mathsf{M}^{+} & \text{if } \Sigma \text{ empty} \\
 & \overset{+}{\rightsquigarrow} & \mathsf{M}^{\div} & \text{if } \Sigma \text{ non-empty}
\end{array}
$$

図 13. DDB の EPS 変換（抜粋）

5.1 EPS 変換後の言語

図 12 に変換後の言語 $d\lambda_{\langle\rangle}$ を示す。この言語は、$\lambda_{\langle\rangle}$ を依存型で拡張したものとして理解される。依存型と部分型が共存する体系では、型保存を保証するために、η に対応する規則を同値性規則として定義することが多いが [36]、$d\lambda_{\langle\rangle}$ では環境のみが部分型付けの対象であり、これらを表す変数には型注釈が与えられないため、η を簡約規則として定義している。

5.2 選択的 EPS 変換

DDB の選択的 EPS 変換は、DB の変換に種と型の規則を追加することで得られる。なお、DDB では、種が型に依存し、型が項に依存するため、これらの変換も導出に関する帰納法によって定義される。図 13 に新しい変換規則を示す。関数型 $\Pi\, \mathsf{x} : \mathsf{A}.\, \mathsf{B}\ \Sigma$ の変換は、依存性をもたない矢印型 $\mathsf{A} \to \mathsf{B}\ \Sigma$ の変換とほぼ同じである。型レベルの関数と関数適用は、項レベルと異なり、1 種類の変換規則をもつ。これは、型が純粋な世界であり、部分項がもつエフェクトの組み合わせで場合分けする必要がないことによる。特に、関数適用 $\mathsf{A}\ \mathsf{M}$ において、引数 M は項であるが、これがエフェクトをもつ場合は $\mathsf{A}\ \mathsf{M}$ が導出できないため、M が純粋であるケースのみを考慮すれば良い。

5.3 EPS 変換の性質

DDB の EPS 変換についても、正当性と型の保存性を証明することができる。単純型付き計算との違いは、型保存の証明が正当性に依存するという点である。これは、⟨A-Conv⟩ と ⟨M-Conv⟩ によって導出された型と項のケースを示す際に、同値性 $K \equiv K'$ と $A \equiv A'$ が変換によって保存されることが必要となるためである。

定理 7 (正当性)**.**

1. $\Gamma \vdash K$ かつ $K \triangleright^* K'$ ならば、$K^+ = K'^+$。
2. $\Gamma \vdash A : K$ かつ $A \triangleright^* A'$ ならば、$A^+ = A'^+$。
3. $\Gamma \vdash_\bullet M : A$ かつ $M \triangleright^* M'$ ならば、$M^+ = M'^+$。
4. $\Gamma \vdash_\Sigma M : A$ かつ $M \triangleright^* M'$ ならば、$M^\div = M'^\div$。

定理 8 (型保存)**.**

1. $\vdash \Gamma \Rightarrow \vdash \Gamma^+$
2. $\Gamma \vdash \Sigma \Rightarrow \Gamma^+ \vdash \Sigma^+ : *$
3. $\Gamma \vdash K \Rightarrow \Gamma^+ \vdash K^+$
4. $\Gamma \vdash A : K \Rightarrow \Gamma^+ \vdash A^+ : K^+$
5. $\Gamma \vdash_\bullet M : A \Rightarrow \Gamma^+ \vdash M^+ : A^+$
6. $\Gamma \vdash_\Sigma M : A \Rightarrow \Gamma^+ \vdash M^\div : \Sigma^+ \to A^+$

Proof. 導出の長さに関する帰納法による。ここでは、以下の導出をもつ関数適用のケースを証明する。

$$\frac{\Gamma \vdash_{\Sigma_0} M : \Pi\, x : A.\, B \quad \Sigma_2 \quad \Gamma \vdash_\bullet N : A \qquad \Gamma \vdash B[N/x] : * \quad \Gamma \vdash \Sigma_2[N/x] \quad \mathsf{consistent}\,(\Sigma_0 \cup \Sigma_2[N/x])}{\Gamma \vdash_{(\Sigma_0 \cup \Sigma_2[N/x])} M\ N : B[N/x]}\ (\text{M-App})$$

この関数適用は $\lambda\, \mathbf{e}.\, (M^\div\ \mathbf{e})\ N^+\ \mathbf{e}$ という関数に変換される。この関数に $(\Sigma_0 \cup \Sigma_2[N/x])^+ \to (B[N/x])^+$ という型を付けたい。帰納法の仮定より、

$$\Gamma^+ \vdash M^\div : {\Sigma_0}^+ \to \mathbf{\Pi}\, \mathbf{x} : A^+.\, {\Sigma_2} \to {}^+B^+ \quad \text{および} \quad \Gamma^+ \vdash N^+ : A^+$$

が成立している。ここで、$(M^\div\ \mathbf{e})\ N^+\ \mathbf{e}$ の型を考えてみよう。まず、$(\Sigma_0 \cup \Sigma_2[N/x])^+ \leq {\Sigma_0}^+$ であるため、関数適用 $M^\div\ \mathbf{e}$ は型安全であり、$\mathbf{\Pi}\, \mathbf{x} : A^+.\, {\Sigma_2}^+ \to B^+$ という型をもつ。これに N^+ を渡すと、${\Sigma_2}^+[N^+/\mathbf{x}] \to B^+[N^+/\mathbf{x}]$ 型の項が得られるが、$(\Sigma_0 \cup \Sigma_2[N/x])^+ \leq (\Sigma_2[N/x])^+$ であり、変換と代入の可換性より $(\Sigma_2[N/x])^+ = {\Sigma_2}^+[N^+/\mathbf{x}]$ が成り立つため、$(M^\div\ \mathbf{e})\ N^+\ \mathbf{e}$ は $B^+[N^+/\mathbf{x}]$ 型をもつ。最後にもう一度可換性を使うと、関数適用全体の型は $(B[N/x])^+$ と等しくなる。よって、$M\ N$ の変換結果は期待された型をもつ。 □

非選択的な変換の場合 DDB に非選択的な EPS 変換を与えた場合、純粋な項も環境を受け取る関数に書き換えられる。しかし、純粋な項は本来環境を必要としないため、その変換結果は特定の型をもつ環境を要求しない。これは、純粋な項について、以下が成立することを意味する（ただし、変換後の言語が多相型をもつことを仮定している）。

$$\Gamma \vdash_\bullet M : A \Rightarrow \Gamma^+ \vdash M^\div : \mathbf{\Pi}\, \alpha : *.\, \alpha \to A^+$$

$\mathsf{M}^{\div}$ の型を見ると、受け取る環境の型が型変数 α で表されている。この α は、M を取り囲む文脈に応じて具体化されるが、ここで代入される型が全称量化を含む場合がある。例として、純粋な項 M と $\mathsf{A} \to \mathsf{B} \bullet$ 型の動的変数 p からなる関数適用 $\mathsf{M}\ \mathsf{p}$ を考えよう。この項に非選択的な EPS 変換を適用すると、$\lambda \mathbf{e}.\, (\mathsf{M}^{\div}\ \mathbf{e})\ (\mathsf{p}^{\div}\ \mathbf{e})\ \mathbf{e}$ という項が得られる。ここで、変換前の項は $\bullet, \mathsf{p} : \mathsf{A} \to \mathsf{B} \bullet$ という注釈をもった型判断によって導出されるため、$\mathbf{e}$ は $(\mathsf{A} \to \mathsf{B} \bullet)^{+} = \mathsf{A}^{+} \to \mathbf{\Pi}\, \alpha : *.\, \alpha \to \mathsf{B}^{+}$ という型の値からなる環境を表す。つまり、$\mathsf{M}^{\div}$ の型に含まれる α は、$\mathsf{M}^{\div}\ \mathbf{e}$ の実行時に $\langle \mathbf{p} : \mathsf{A}^{+} \to \mathbf{\Pi}\, \alpha : *.\, \alpha \to \mathsf{B}^{+} \rangle$ という多相型に具体化される。これは、純粋な項に対する変換が非叙述的な多相を要求することを意味する。非叙述性は、型の階層や古典論理の公理と組み合わせた際に矛盾を導くため [14, 13, 23]、定理証明の目的で使われる依存型付き言語では、その使用が制限されている場合が多い。したがって、DDB から動的変数を除去するときには、非叙述性を必要としない選択的な変換を用いた方が望ましい。

6 応用例

本研究では、依存型付き言語にプロンプトタグ付き限定継続命令を導入するという目標に向けて DDB を設計したが、この言語自体も、特定の変数の値に柔軟性をもたせながら、意図通りの動作を静的に保証することを可能にする。本節では、DDB ですぐに実現できるプログラムと、より実用的な応用例を紹介する。

6.1 リストの生成

DDB を再帰関数と $\mathsf{L}\ \mathsf{n}$ 型のリストで拡張すると、次のような関数を定義できる [2]。

$$\mathsf{rec\ f} : (\Pi\, \mathsf{x} : \mathbb{N}.\, \mathsf{L\ x}\ (\bullet, \mathsf{p} : \mathbb{N})).\, \mathsf{x}.\, \mathsf{match\ x\ with\ z} \to \mathsf{nil} \mid \mathsf{suc\ n} \to \mathsf{cons\ p\ (f\ n)}$$

関数 f は、自然数 x を受け取ると、x 個の p からなるリストを返す。リスト型に長さのインデックスをもたせることにより、単純型付きの実装と比べて、結果のリストが満たす性質をより正確に表現できていることが分かる。一方、リストの要素は動的変数 p で表されているため、出力されるリストの中身は、f を呼び出す際の評価文脈に依存する。たとえば、f を $\mathsf{dlet\ p} = 1 : \mathbb{N}\ \mathsf{in}\ [\,]$ という文脈の中で呼び出した場合、リストの中身はすべて 1 となる。

$$\mathsf{dlet\ p} = 1 : \mathbb{N}\ \mathsf{in\ f\ 3} = \mathsf{cons\ 1\ (cons\ 1\ (cons\ 1\ nil))}$$

上の関数がもつ文脈への依存性が問題とならないのは、p が型に現れない形で使用されているからである。もし、p が f に渡されていた場合、$\langle \textsc{App} \rangle$ 規則の前提（$\Sigma_1 = \bullet$）が満たされないため、プログラム全体に型が付かなくなる。

6.2 Pretty Printer

DDB の実世界における応用例として、pretty printer が挙げられる。pretty printer は、ユーザのソースコードを受け取ったら、インデントや改行を挿入することで、コードを読みやすい形に加工する [20]。ここで、加工後のコードの幅がユーザのエディタの幅を越えないようにするためには、幅の最大値をプリンタの実装時ではなく、プリンタが呼び出されたときに定める必要がある。このような束縛は、幅を動的変数で表現することで容易に実現できる [25]。さらに、依存型を使うと、

[2] 厳密には、2 つのブランチがもつエフェクトを揃えるために、純粋な項をそうでない項へと変換する型規則が必要となる。

出力の文法的な正しさに関する証明が付いたプリンタの実装が可能となる [8]。アイディアとしては、加工後のコードの型を元のコードとその文法に依存させることによって、プリンタを適用する前後のコードが同一であることを保証する。このように、動的変数と依存型を用いると、関数が受け取る引数を最小限に抑えながら、正しいプリンタを実装することができる。

6.3 分散プログラミング

もう 1 つの応用先として、分散プログラミングが考えられる。分散システムでは、あるプログラムを実行する際に、プログラム自体を別のマシンに移動させることがある。ここで、受け渡されるプログラムが現在のホスト名を参照する場合、その値は実行時の環境に依存するため、プログラムの生成時に固定されてはならない。動的束縛をサポートする言語では、ホスト名を動的変数で表すことによって、期待された束縛を実現できる [27, 24]。一方で、依存型を用いると、クライアント間のやり取りが安全に行われることを保証できる。たとえば、料金決済システムの中で、悪意のあるウェブサイトが「決済の失敗」という偽のメッセージを表示して料金をだまし取ることがあるが、これは、依存型を使って、表示されるメッセージと、各クライアントの状態が矛盾しないことを保証することで回避される [34]。このように、動的変数と依存型を組み合わせた言語は、安全な分散システムの簡潔な実装につながると考えられる。

7 関連研究

Moreau [27] は、動的束縛をもつ計算体系の公理化を行っている。導出された公理は、動的変数を含むプログラムで成立する同値関係を表しており、EPS 変換に関して健全かつ完全であることが示されている。Moreau はさらに、環境の受け渡しによる計算ステップがより少ない EPS 変換を定義し、これを用いて、評価関数の正当性を証明している。この変換は選択的でないが、定義の中で部分項が値であるかどうかを判断するという点において、本研究の選択的な変換と類似している。

動的束縛をサポートするためのアプローチとして、Lewis ら [25] は Haskell を非明示的変数（implicit parameter）で拡張している。非明示的変数は、プログラマによって明示的に宣言されるのではなく、プログラム中の出現に基づいて、その存在が推論される。これによって、動的変数を用いた場合と同様に、グローバルな変数の使用や、個々の関数に渡す引数の追加を避けることができる。非明示的変数と動的変数の主な違いは、前者を含む関数が第一級の値とみなされず、他の関数に引数として渡されることがないという点である。この制約は、動的変数が引き起こす downward funarg 問題を回避するために設けられている。

動的束縛と類似した機能を実現するために、Hashimoto and Ohori [17] は文脈を第一級のオブジェクトとして扱うことのできる計算体系を提案している。彼らの主な貢献は、文脈と項を合成させたときに発生する静的変数のキャプチャを許しながら、合流性（confluence）を保持するための手法を与えたことである。DDB では、動的変数のみがキャプチャされ得るため、合流性を示すのはそれほど難しくない。一方、Hashimoto and Ohori の型システムは、自由な hole が複数回現れないように設計されており、この性質は、動的変数の環境が重複した名前を含まないという DDB の性質と対応している。

Kiselyov ら [24] は、動的束縛と限定継続がインタラクトする計算体系を築いている。彼らは、動的変数の値が、限定継続命令の切り取る継続と同じように、自身を取り囲む文脈によって定まるという点に着目し、動的束縛のメカニズムを限定継続で表現できることを示した。しかし、Kiselyov らの言語では、型システムから項がもつエフェクトの情報が得られないため、選択的な変換の定義や、依存型による拡張には向いていない。

動的束縛と限定継続を扱うもう 1 つの研究として、Downen & Ariola [11] は、プロンプトタグ付き $\lambda\mu\tilde{\mu}$ 計算 [7] の意味論を与えている。興味深いことに、彼らは動的変数を使って限定継続の意味

を表現するという、Kiselyov らと逆のアプローチをとっている。具体的には、限定継続命令を除去する際に、CPS 変換によってプロンプトタグを動的変数に置き換え、さらに EPS 変換を適用することで、純粋な λ 計算のプログラムを得ている。Downen & Ariola の変換は型なしの言語に対して定義されているが、変換前の言語において型の依存性を適切に制限すれば、これらの変換を依存型付きのものに拡張できると考えられる。

動的変数は、代数的エフェクトを用いて表現することも可能である。Kammar and Pretnar [22] は、Kiselyov らの言語と類似した動的変数付き λ 計算から、Plotkin and Pretnar [30] のエフェクト言語への変換を与えている。アイディアとしては、各動的変数 p に対して、その値を参照・更新する演算 get_p, set_p を定義し、限定継続命令による state モナドの実装 [1] と類似したハンドラによって、これらの演算に意味を与える。代数的エフェクトをもつ言語では、関数型にエフェクトの注釈 Σ を付与することが多いが、大域的な環境を型にもたせると、型と環境の変換が well-founded でなくなってしまう。そこで、Kammar and Pretnar は、元のプログラムが高階の動的変数を使わないときにのみ、変換後の関数型にエフェクトの情報をもたせている。なお、代数的エフェクトを含むプログラムは、CPS 変換によって純粋なプログラムに書き換えられることが知られているが [19]、型付きの変換が可能であるかどうかはまだ明らかにされていない。

8 結論

本稿では、動的変数をもつ依存型付き言語 DDB を設計した。DDB では、動的変数への参照をエフェクトと捉え、型規則にエフェクトの情報をもたせた。これを用いて、型の依存性を純粋な項に制限するとともに、非叙述性を仮定しない EPS 変換を定義した。依存性を純粋な項に制限するというのは、過去の研究でもとられているアプローチだが、選択的な変換によってエフェクトを除去するというアイディアも、依存型とエフェクトを組み合わせる際に広く適用できると考えられる。

今後は、DDB と類似した手法で、プロンプトタグ付き `shift`/`reset` をもつ依存型付き言語と、それに対する CPS 変換を定義する予定である。CPS 変換後のプログラムは DDB のサブセットにおさまるため、本稿で与えた EPS 変換を適用すれば、純粋な依存型付き λ 計算のプログラムが得られるはずである。出来上がった言語の応用例としては、型付き言語に対する部分評価器 [9] や、複雑なデータに対する探索プログラムなどが考えられる。

謝辞

型システムの設計や関連研究の議論に関して、多くのアドバイスをくださった査読者のみなさまに深く感謝いたします。本研究は一部 JSPS 科研費 18H03218 の助成を受けたものです。

参考文献

[1] Kenichi Asai and Oleg Kiselyov. Introduction to programming with shift and reset. In *ACM SIGPLAN Continuation Workshop*, September 2011.

[2] Kenichi Asai and Chihiro Uehara. Selective CPS transformation for shift and reset. In *Proceedings of the 2018 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation*, PEPM '18, pages 40–52, 2018.

[3] L. Cardelli. Structural subtyping and the notion of power type. In *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '88, pages 70–79, New York, NY, USA, 1988. ACM.

[4] Chris Casinghino, Vilhelm Sjöberg, and Stephanie Weirich. Combining proofs and programs in a dependently typed language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 33–45, New York, NY, USA, 2014. ACM.

[5] Youyou Cong and Kenichi Asai. Handling delimited continuations with dependent types. *Proc. ACM Program. Lang.*, 2(ICFP):69:1–69:31, September 2018.

[6] Thierry Coquand. An analysis of girard's paradox. In *Symposium on Logic in Computer Science (LICS)*, pages 227–236, 1986.

[7] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *ACM sigplan notices*, volume 35, pages 233–243. ACM, 2000.

[8] Nils Anders Danielsson. Correct-by-construction pretty-printing. In *Proceedings of the 2013 ACM SIGPLAN Workshop on Dependently-typed Programming*, pages 1–12. ACM, 2013.

[9] Olivier Danvy. Type-directed partial evaluation. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '96, pages 242–257, New York, NY, USA, 1996. ACM.

[10] Olivier Danvy and Andrzej Filinski. Abstracting control. In *Proceedings of the 1990 ACM conference on LISP and functional programming*, pages 151–160. ACM, 1990.

[11] Paul Downen and Zena M Ariola. Delimited control and computational effects. *Journal of functional programming*, 24(1):1–55, 2014.

[12] Andrzej Filinski. Representing monads. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, pages 446–457, New York, NY, USA, 1994. ACM.

[13] Herman Geuvers. Inconsistency of classical logic in type theory, 2001. Unpublished notes.

[14] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique*. PhD thesis, Université Paris VII, 1972.

[15] Carl A. Gunter, Didier Rémy, and Jon G. Riecke. A generalization of exceptions and control in ml-like languages. In *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture*, FPCA '95, pages 12–23, New York, NY, USA, 1995. ACM.

[16] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993.

[17] Masatomo Hashimoto and Atsushi Ohori. A typed context calculus. *Theor. Comput. Sci.*, 266(1-2):249–272, September 2001.

[18] Hugo Herbelin. On the degeneracy of Σ-types in presence of computational classical logic. In *International Conference on Typed Lambda Calculi and Applications*, TLCA '05, pages 209–220. Springer, 2005.

[19] Daniel Hillerström, Sam Lindley, Robert Atkey, and KC Sivaramakrishnan. Continuation passing style for effect handlers. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, 2017.

[20] John Hughes. The design of a pretty-printing library. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques-Tutorial Text*, pages 53–96, Berlin, Heidelberg, 1995. Springer-Verlag.

[21] Yukiyoshi Kameyama and Masahito Hasegawa. A sound and complete axiomatization of delimited continuations. In *Proceedings of the 8th ACM SIGPLAN International Conference on Functional Programmin*, ICFP '03, pages 177–188, New York, NY, USA, 2003. ACM.

[22] Ohad Kammar and Matija Pretnar. No value restriction is needed for algebraic effects and handlers. *Journal of Functional Programming*, 27, 2017.

[23] Chantal Keller and Marc Lasson. Parametricity in an impredicative sort. In *CSL*, volume 16, pages 381–395. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2012.

[24] Oleg Kiselyov, Chung-chieh Shan, and Amr Sabry. Delimited dynamic binding. In *The 33th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '06, pages 26–37. ACM, 2006.

[25] Jeffrey R. Lewis, John Launchbury, Erik Meijer, and Mark B. Shields. Implicit parameters: Dynamic scoping with static types. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, pages 108–118, New York, NY, USA, 2000. ACM.

[26] Étienne Miquey. A classical sequent calculus with dependent types. In *European Symposium on Programming*, pages 777–803. Springer, 2017.

[27] Luc Moreau. A syntactic theory of dynamic binding. *Higher-Order and Symbolic Computation*, 11(3):233–279, 1998.

[28] Lasse R Nielsen. A selective cps transformation. *Electronic Notes in Theoretical Computer Science*, 45:311–331, 2001.

[29] Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers University of Technology, 2007.

[30] Gordon Plotkin and Matija Pretnar. Handlers of algebraic effects. In *European Symposium on Programming*, ESOP '09, pages 80–94. Springer, 2009.

[31] Tiark Rompf, Ingo Maier, and Martin Odersky. Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform. In *Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming*, ICFP '09, pages 317–328. ACM, 2009.

[32] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and symbolic computation*, 6(3-4):289–360, 1993.

[33] Vilhelm Sjöberg, Chris Casinghino, Nathan Collins, Yung Ki Ahn, Tim Sheard, Harley D Eades III, Peng Fu, Garrin Kimmell, Aaron Stump, and Stephanie Weirich. Irrelevance, heterogeneous equity, and call-by-value dependent type systems. In *Fourth Workshop on Mathematically Structured Functional Programming*, MSEP '12, 2012.

[34] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. In *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming*, ICFP '11, pages 266–278, New York, NY, USA, 2011. ACM.

[35] Masako Takahashi. Parallel reductions in λ-calculus. *Information and computation*, 118(1):120–127, 1995.

[36] The Coq Development Team. The Coq proof assistant reference manual. `https://coq.inria.fr/refman/`, March 2018.

[37] Sergei Vorobyov. Subtyping functional+nonempty record types. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, *Computer Science Logic*, pages 283–297, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[38] Andrew K Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and computation*, 115(1):38–94, 1994.

A　DB の評価文脈の型付け

$$\frac{}{\Gamma \vdash [] : A \to A \bullet}\ \langle \text{E-Hole} \rangle$$

$$\frac{\Gamma \vdash E[] : B \to A' \ \Sigma \quad \Gamma \vdash_{\Sigma_1} N : A \quad \text{consistent}\ ((\Sigma_1 \cup \Sigma_2 \setminus \mathrm{BP}\,(E[])) \cup \Sigma)}{\Gamma \vdash E[[]\ N] : (A \to B\ \Sigma_2) \to A'\ ((\Sigma_1 \cup \Sigma_2 \setminus \mathrm{BP}\,(E[])) \cup \Sigma)}\ \langle \text{E-App1} \rangle$$

$$\frac{\Gamma \vdash E[] : B \to A' \ \Sigma \quad \Gamma \vdash_\bullet V : A \to B\ \Sigma_2 \quad \text{consistent}\ ((\Sigma_2 \setminus \mathrm{BP}\,(E[])) \cup \Sigma)}{\Gamma \vdash E[V\ []] : A \to A'\ ((\Sigma_2 \setminus \mathrm{BP}\,(E[])) \cup \Sigma)}\ \langle \text{E-App2} \rangle$$

$$\frac{\Gamma \vdash E[] : B \to A' \ \Sigma \quad \Gamma \vdash_\bullet V : A}{\Gamma \vdash E[\mathsf{dlet}\ p = V : A\ \mathsf{in}\ []] : B \to A'\ \Sigma}\ \langle \text{E-DLet} \rangle$$

B　評価文脈の EPS 変換

$\langle$E-Hole$\rangle$ $\overset{+}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : A^+ .\, \mathbf{x}$

$\langle$E-App1$\rangle$ $\overset{+}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \bullet)^+ .\, E[]^+\ (\mathbf{x}\ N^+)$
if $\Sigma_1, \Sigma_2, \Sigma$ empty

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \bullet)^+ .\, \lambda\, \mathbf{e}.\, E[]^+\ (\mathbf{x}\ (N^\div\ \mathbf{e}'))$
if Σ_2, Σ empty, Σ_1 non-empty; $\mathbf{e}' = \langle \mathbf{e}, \text{build-env}(E[], \Sigma_1) \rangle$

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \Sigma_2)^+ .\, \lambda\, \mathbf{e}.\, E[]^+\ (\mathbf{x}\ N^+\ \mathbf{e}')$
if Σ_1, Σ empty, Σ_2 non-empty; $\mathbf{e}' = \langle \mathbf{e}, \text{build-env}(E[], \Sigma_2) \rangle$

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \Sigma_2)^+ .\, \lambda\, \mathbf{e}.\, E[]^+\ (\mathbf{x}\ (N^\div\ \mathbf{e}')\ \mathbf{e}')$
if Σ empty, Σ_1, Σ_2 non-empty; $\mathbf{e}' = \langle \mathbf{e}, \text{build-env}(E[], \Sigma_1 \cup \Sigma_2) \rangle$

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \bullet)^+ .\, \lambda\, \mathbf{e}.\, E[]^\div\ (\mathbf{x}\ N^+)\ \mathbf{e}$
if Σ_1, Σ_2 empty, Σ non-empty

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \bullet)^+ .\, \lambda\, \mathbf{e}.\, E[]^\div\ (\mathbf{x}\ (N^\div\ \mathbf{e}'))\ \mathbf{e}$
if Σ_2 empty, Σ_1, Σ non-empty; $\mathbf{e}' = \langle \mathbf{e}, \text{build-env}(E[], \Sigma_1) \rangle$

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \Sigma_2)^+ .\, \lambda\, \mathbf{e}.\, E[]^\div\ (\mathbf{x}\ N^+\ \mathbf{e}')\ \mathbf{e}$
if Σ_1 empty, Σ_2, Σ non-empty; $\mathbf{e}' = \langle \mathbf{e}, \text{build-env}(E[], \Sigma_2) \rangle$

$\overset{\div}{\rightsquigarrow}$ $\lambda\, \mathbf{x} : (A \to B\ \Sigma_2)^+ .\, \lambda\, \mathbf{e}.\, E[]^\div\ (\mathbf{x}\ (N^\div\ \mathbf{e}')\ \mathbf{e}')\ \mathbf{e}'$
if $\Sigma_1, \Sigma_2, \Sigma$ non-empty; $\mathbf{e}' = \langle \mathbf{e}, \text{build-env}(E[], \Sigma_1 \cup \Sigma_2) \rangle$

$$
\begin{aligned}
\langle \text{E-App2} \rangle \quad & \overset{+}{\rightsquigarrow} \quad \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \mathsf{E}[\,]^{+}\ (\mathsf{V}^{+}\ \mathbf{x}) \\
& \qquad \text{if } \Sigma_2, \Sigma \text{ empty} \\
& \overset{\div}{\rightsquigarrow} \quad \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^{+}\ (\mathsf{V}^{+}\ \mathbf{x}\ \mathbf{e}') \\
& \qquad \text{if } \Sigma \text{ empty, } \Sigma_2 \text{ non-empty; } \mathbf{e}' = \langle \mathbf{e}, \text{build-env}(\mathsf{E}[\,], \Sigma_2) \rangle \\
& \overset{\div}{\rightsquigarrow} \quad \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^{\div}\ (\mathsf{V}^{+}\ \mathbf{x})\ \mathbf{e} \\
& \qquad \text{if } \Sigma_2 \text{ empty, } \Sigma \text{ non-empty} \\
& \overset{\div}{\rightsquigarrow} \quad \lambda\, \mathbf{x} : \mathsf{A}^{+}.\, \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^{\div}\ (\mathsf{V}^{+}\ \mathbf{x}\ \mathbf{e}')\ \mathbf{e} \\
& \qquad \text{if } \Sigma_2, \Sigma \text{ non-empty; } \mathbf{e}' = \langle \mathbf{e}, \text{build-env}(\mathsf{E}[\,], \Sigma_2) \rangle \\
\langle \text{E-DLet} \rangle \quad & \overset{+}{\rightsquigarrow} \quad \lambda\, \mathbf{x} : \mathsf{B}^{+}.\, \mathsf{E}[\,]^{+}\ \mathbf{x} \\
& \qquad \text{if } \Sigma \text{ empty} \\
& \overset{\div}{\rightsquigarrow} \quad \lambda\, \mathbf{x} : \mathsf{B}^{+}.\, \lambda\, \mathbf{e}.\, \mathsf{E}[\,]^{\div}\ \mathbf{x}\ \mathbf{e} \\
& \qquad \text{if } \Sigma \text{ non-empty}
\end{aligned}
$$

C DDB の Parallel Reduction

$$\frac{}{\mathsf{T} \rhd_p \mathsf{T}}\ \langle \text{PT-Refl} \rangle \qquad \frac{\mathsf{A} \rhd_p \mathsf{A}' \quad \mathsf{T} \rhd_p \mathsf{T}' \quad \Sigma \rhd_p \Sigma'}{\Pi\, \mathsf{x} : \mathsf{A}.\, \mathsf{T}\ \Sigma \rhd_p \Pi\, \mathsf{x} : \mathsf{A}'.\, \mathsf{T}'\ \Sigma'}\ \langle \text{PT-Pi} \rangle$$

$$\frac{\mathsf{A} \rhd_p \mathsf{A}' \quad \mathsf{T} \rhd_p \mathsf{T}'}{\lambda\, \mathsf{x} : \mathsf{A}.\, \mathsf{T} \rhd_p \lambda\, \mathsf{x} : \mathsf{A}'.\, \mathsf{T}'}\ \langle \text{PT-Abs} \rangle$$

$$\frac{\mathsf{T} \rhd_p \mathsf{T}' \quad \mathsf{M} \rhd_p \mathsf{M}'}{\mathsf{T}\ \mathsf{M} \rhd_p \mathsf{T}'\ \mathsf{M}'}\ \langle \text{PT-App} \rangle \qquad \frac{\mathsf{T} \rhd_p \mathsf{T}' \quad \mathsf{V} \rhd_p \mathsf{V}'}{(\lambda\, \mathsf{x} : \mathsf{A}.\, \mathsf{T})\ \mathsf{V} \rhd_p \mathsf{T}'[\mathsf{V}'/\mathsf{x}]}\ \langle \text{PT-Beta} \rangle$$

$$\frac{\mathsf{V} \rhd_p \mathsf{V}' \quad \mathsf{A} \rhd_p \mathsf{A}' \quad \mathsf{M} \rhd_p \mathsf{M}'}{\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ \mathsf{M} \rhd_p \mathsf{dlet}\ \mathsf{p} = \mathsf{V}' : \mathsf{A}'\ \mathsf{in}\ \mathsf{M}'}\ \langle \text{PT-DLet} \rangle$$

$$\frac{\mathsf{V} \rhd_p \mathsf{V}' \quad \mathsf{E}[\,] \rhd_p \mathsf{E}'[\,]}{\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ \mathsf{E}[\mathsf{p}] \rhd_p \mathsf{dlet}\ \mathsf{p} = \mathsf{V}' : \mathsf{A}'\ \mathsf{in}\ \mathsf{E}'[\mathsf{V}']}\ \langle \text{PT-DLet1} \rangle$$

$$\frac{\mathsf{V}_1 \rhd_p \mathsf{V}_1'}{\mathsf{dlet}\ \mathsf{p} = \mathsf{V}_0 : \mathsf{A}\ \mathsf{in}\ \mathsf{V}_1 \rhd_p \mathsf{V}_1'}\ \langle \text{PT-DLet2} \rangle$$

$$\frac{}{\mathsf{E}[\,] \rhd_p \mathsf{E}[\,]}\ \langle \text{PE-Refl} \rangle \qquad \frac{\mathsf{E}[\,] \rhd_p \mathsf{E}'[\,] \quad \mathsf{M} \rhd_p \mathsf{M}'}{\mathsf{E}[[\,]\ \mathsf{M}] \rhd_p \mathsf{E}'[[\,]\ \mathsf{M}']}\ \langle \text{PE-App1} \rangle$$

$$\frac{\mathsf{E}[\,] \rhd_p \mathsf{E}'[\,] \quad \mathsf{V} \rhd_p \mathsf{V}'}{\mathsf{E}[\mathsf{V}\ [\,]] \rhd_p \mathsf{E}'[\mathsf{V}'\ [\,]]}\ \langle \text{PE-App2} \rangle \qquad \frac{\mathsf{E}[\,] \rhd_p \mathsf{E}'[\,] \quad \mathsf{V} \rhd_p \mathsf{V}' \quad \mathsf{A} \rhd_p \mathsf{A}'}{\mathsf{E}[\mathsf{dlet}\ \mathsf{p} = \mathsf{V} : \mathsf{A}\ \mathsf{in}\ [\,]] \rhd_p \mathsf{E}'[\mathsf{dlet}\ \mathsf{p} = \mathsf{V}' : \mathsf{A}'\ \mathsf{in}\ [\,]]}\ \langle \text{PE-DLet} \rangle$$