2019 年度 修士論文

版を扱えるプログラミング言語 λ_{VL} への型・版多相性の導入の研究

東京工業大学大学院 情報理工学院
数理・計算科学専攻

学籍番号 18M30226

田辺 裕大

指導教員

増原 英彦 教授

2020 年 1 月 23 日

概要

本研究では言語要素として版の概念を取り込んだプログラミング言語 λ_{VL} の型システムに、型多相性と版多相性を導入する。λ_{VL} は、版によって実装の有無が変わる関数を使用したプログラムを許容し、そのようなプログラムが版において矛盾のない計算をすることを保証する。更に複数の版で無矛盾なプログラムは、必要に応じて適切な版を選択出来る。型多相性の導入によってモジュール機構が提供可能になり、版多相性によって特定の版に依存しないプログラムが書けるようになる。本研究は、２つの多相性の導入をカインドに基づく型システムの拡張によって行い、意味論を与え、健全性を証明した。

謝辞

本研究は 2017 年夏ごろの学部卒業研究から始まり、途中 1 年の中断 (2018/11 から 2019/11 まで) を挟んで 2 年半に渡り検討を重ねてきたものです。青谷知幸さんには研究の初期の方向性の指導や参考文献を頂き、共に多大なる有益な議論を重ねました。助教の叢悠悠先生には定期的に本論文のレビューを行っていただきました。関連する共同研究として、オブジェクト指向言語へのバージョンの概念の導入は Lubis Luthfan 君と共に検討を重ねました。最後に、指導教官の増原英彦先生には本研究のユースケースや大枠について沢山の助言を頂き、λ_{VL} 論文の論文投稿・海外発表の機会を頂きました。以上の方々に多大なる感謝の意を表します。

目次

第 1 章

はじめに

1.1 モジュール性とライブラリ

エンジニアリングに於けるモジュール性とは、対象が機能・目的に従って適切に分割されていることを指す性質である。モジュール性はソフトウェア分野においても重要な概念である。良いモジュール性を持つプログラムは開発・保守が容易であることが知られており、開発対象の大きさ・複雑さが増大するに連れてモジュール性は一層重要になる。またプログラムをモジュール性に従って分割したものを**ライブラリ**と呼ぶ。ライブラリ単位でプログラムを製作・公開し、また同時に他人の公開した有用ライブラリを使用して開発を進めることは現代のソフトウェア開発のスタンダードである。

ライブラリで使用されている関数や値をグループ化するプログラミング言語機構が**モジュール (パッケージ) 機構**である。モジュール機構の重要な役割の一つは、**内部の実装と外部に公開される仕様の分離**である。これによりライブラリの使用者は開発プログラムと依存ライブラリの実装を分離でき、ライブラリの設計者はそのライブラリの実装を隠蔽・抽象化することでライブラリの使途を制限可能になるという利点がある。言語毎に実現方法は異なるが、近年ソフトウェア開発で使用されるプログラミング言語のほぼ全てがモジュール機構を備えており、大規模プログラミングやソフトウェアのモジュール化を支援している。

1.2 バージョン違いの非互換性と依存性地獄

多くのライブラリには機能改善やバグ修正を目的としたアップデートが入るため、複数のバージョンが存在する。このため同名のライブラリであっても、しばしばバージョン違いで非互換性が発生する。多くのアプリケーションは各ライブラリの特定のバージョンの仕様に依存しているため、アップデート等によりアプリケーションの実行に必要なバージョンと異なるバージョンのライブラリが提供されると、2 つのバージョンの差異によってはアプリケーションが正しく動作しない場合がある。

この問題はライブラリの依存性が引き起こす推移的な依存関係においてより複雑となり、これらの問題群は**依存性地獄**として広く認知されている。現代の標準的な依存性地獄への対応は、パッケージマネージャ等を用いて各ライブラリのバージョンを厳密に管理することである。本研究ではこのアプローチとは異なり、プログラミング言語の中でバージョンを扱う仕組みを検討する。

1.3 値レベルのバージョン依存性

動的型付け言語における依存性地獄は、モジュールによって階層化された名前空間上でのシンボルの衝突問題と同一である。したがって最も簡単な解決策は、両者のライブラリを全く別のものとして扱ってしまうことである。多くのプログラミング言語ではモジュール名に名前空間上での別名をつける仕組みを有しており、バージョン違いのライブラリに別名をつけることで、明示的に各バージョンの実装を区別して使うことが可能となる。

一方で静的型付け言語の立場では、バージョン違いを全く別のライブラリとして扱う手法には問題がある。第一の問題は、全てのバージョン違いのプログラムが互いの関係性を失ってしまうことだ。多くのライブラリのアップデートにおいて非互換な変更は局所的であり、バージョン違いのライブラリの大部分は互換性を保っている。また、変更の行われたプログラムにおいても、その一部は前のバージョンとの後方互換性が存在する。これら実際には区別する必要のないプログラムを全く別のものとして扱ってしまうのは合理的ではない。第二の問題は、モジュール機構の原則に違反していることである。先に述べた通り、ライブラリの実装を抽象化するのは原則的にインターフェースの役割であり、使用者側のプログラムでバージョン違いの吸収を行うのは合理的ではない。プログラミング言語の観点からすると、これらの問題は値レベルでバージョン間のプログラムの関係を記述する仕組みがない、すなわち**バージョンに関する情報をキャプチャする言語機構が存在しない**ことに起因する。

1.4 本研究の貢献

本研究では静的型付け言語における依存性地獄を解決するため、バージョンの概念を取り込んだプログラミング言語 λ_{VL}[1] を提案する。さらに、λ_{VL} にバージョン抽象・モジュール機構を導入するための準備段階として、型・版多相性の導入手法を提案する。

1.4.1 λ_{VL}

λ_{VL} は以下の機能を持つ。

- プログラムに複数のバージョンの定義を持たせることが可能
- プログラムの各バージョンの定義を区別して参照可能
- プログラムの実行に必要なライブラリとそのバージョンを静的に解析可能

これらの機能は、同一環境にバージョン違いのライブラリを共存させ、同名のライブラリを区別するためのプログラミング言語的機構を提供する。バージョンによって実装の有無が変わる関数を使用したプログラムを許容し、そのようなプログラムがバージョンにおいて矛盾のない計算をすることを保証する。さらに、複数のバージョンにおいて無矛盾なプログラムは、必要に応じて適切なバージョンを選択することができる。

1.4.2 型・版多相性の導入

本研究では単純型付きラムダ計算をベースに作られたコア部分の λ_{VL} の型システムに、型多相性と版多相性を導入した。型多相性の導入によってモジュール機構やジェネリクスが提供可能になり、版多相性の導入によって特定のバージョンに依存しないプログラムが書けるようになる。本研究は 2 つの多相性の導入をカインドに

基づく λ_{VL} の型システムの拡張によって行い、意味論を与え、健全性を証明した。

1.5 本論文の構成

第 2 章でライブラリのバージョンに関する問題を論ずる。第 3 章で λ_{VL} の言語機構を紹介し、第 2 章で論じた問題の解決手法を解説する。第 4 章で本研究により拡張された λ_{VL} の構文と型システムについて紹介する。さらに新たに与えられた意味論を解説し、それに基づく型安全性について議論する。最後に、第 5 章で関連研究を紹介し、第 6 章で今後の言語拡張の方向性について述べる。

第 2 章

バージョンに関する問題

2.1 バージョン違いの非互換性

バージョン違いによる非互換性には、大別して**構造的な非互換性**と**動作的な非互換性**がある。本節ではこれら二つの非互換性について Android API[2] を例にとって説明する。なお、Android API ではバージョンとして API レベルという呼称を用いており、以降これに従うことに注意する。

構造的な非互換性とは、二つのバージョンのライブラリでクラス・データ構造・関数などの**インターフェース上での表現が異なる状況**の事である。例として Android API の `AndroidHttpClient` クラスを挙げる（表 2.1）。`AndroidHttpClient` クラスは Android OS 上での http 接続の為に提供されていたクラスである。しかし、Android 6.0 での Apache HTTP クライアントへのサポート終了に合わせ、API レベル 22 での非推奨期間を経て、API レベル 23 のアップデートにより削除された。API レベル 23 以降の Anroid OS 上での http 接続は、Open JDK 8 ベースの実装である `HttpURLConnection` クラスを使う必要がある。このように、特にサポートを終了する機能について、バージョン違いでインターフェースが異なることがある。これが構造的な非互換性の一例である。

動作的な非互換性とは、二つのバージョンのライブラリでクラス・データ構造・関数などの**インターフェース上では変更がないにも関わらず、その内部実装が異なっている状況**のことである。例として Android API のクラス `AlarmManager` に定義されたメソッド `set` を挙げる（表 2.2）。API レベル 18 以前の `set` は特定の時刻を表す `long` 型の値を取り、その時刻にアラームを鳴らすトリガーを設定することができる。しかし API レベル 19 のアップデートで、端末の電力効率を高めることを目的として、`AlarmManager` のスケジューリング戦略が変更

API レベル	8〜21	22	23〜
`AndroidHttpClient`	サポート有	非推奨	削除
`HttpURLConnection`	サポート有	サポート有	サポート有

表 2.1 Android での http 通信のサポート状況

API レベル	〜18	19〜
`AlarmManager.set`	正確なアラーム	非正確なアラーム
`AlarmManager.setExact`	サポート無	正確なアラーム

表 2.2 Android でのアラームスケジューリング

された。レベル 19 以後、同時期に AlarmManager.set でトリガーされたアラームは纏められて一度に着火するようになり、場合によっては着火時刻に数秒単位のズレが発生するようになった。API レベル 18 以前の set と同様の正確なアラームスケジューリングを実現するためには、新しいメソッド setExact を使う必要がある。このように、特にプラットフォームの機能のアップデートに合わせ、同じ名前のメソッドとして定義されているにも関わらず、バージョン違いで実装が異なるプログラムが存在する。これが動作的な非互換性の一例である。

2.2 依存性地獄

前節で述べたバージョン違いによる非互換性は、ライブラリの機能アップデートや、設計方針の変更などで不可避的に発生するものである。バージョン違いによる非互換性は、大別して**アプリケーションの誤動作**と、**バージョン違いのライブラリの共存不可**という 2 つの問題を発生させる。これらの問題は**依存性地獄**として広く認知されている。

2.2.1 第一の問題：アプリケーションの誤動作

第一の問題は、非互換性により引き起こされるアプリケーションの誤動作である。多くのアプリケーションは各ライブラリの特定のバージョンに依存しているため、アプリケーションの実行に必要なバージョンと異なる（特により新しい）バージョンのライブラリが提供されると、2 つのバージョンの差異によってはアプリケーションが正しく動作しない場合がある。

例えば構造的な非互換性について、AndroidHttpClient クラスを用いて通信機能を実装したアプリケーションがあるとする（図 2.1）。Android OS 6.0 へのアップデートに伴い、共有ライブラリの API レベルが 23 にアップデートされた状況を考える。前節で述べたように、このアップデートにより AndroidHttpClient クラスは共有ライブラリから削除される。アプリケーションの製作者が Android API 23 への対応を行わなかった場合、ア

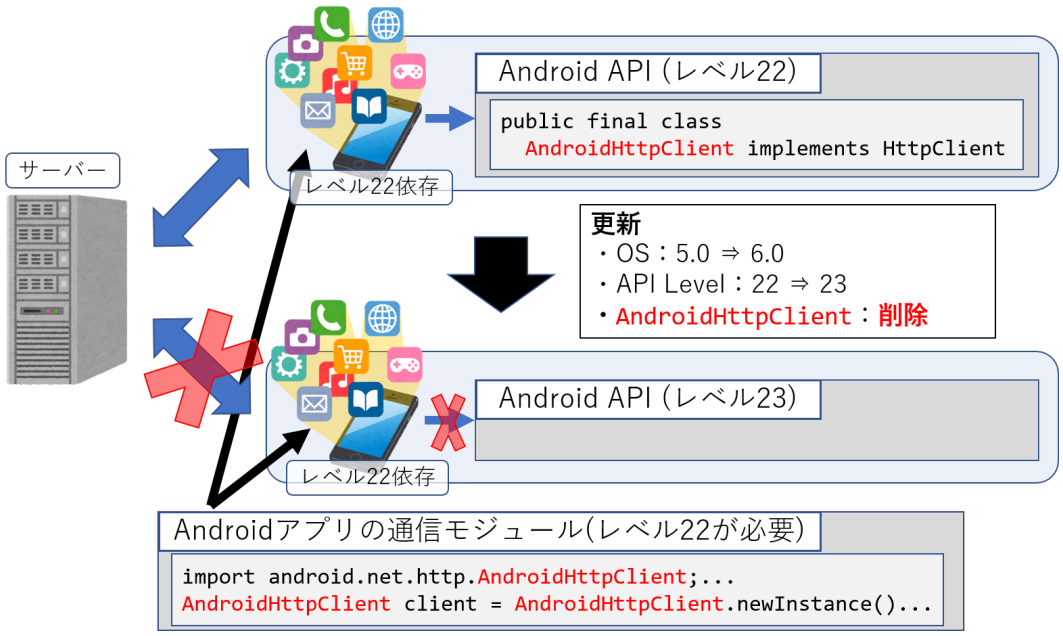

図 2.1 共有ライブラリのアップデートが原因で通信に失敗する Android アプリ

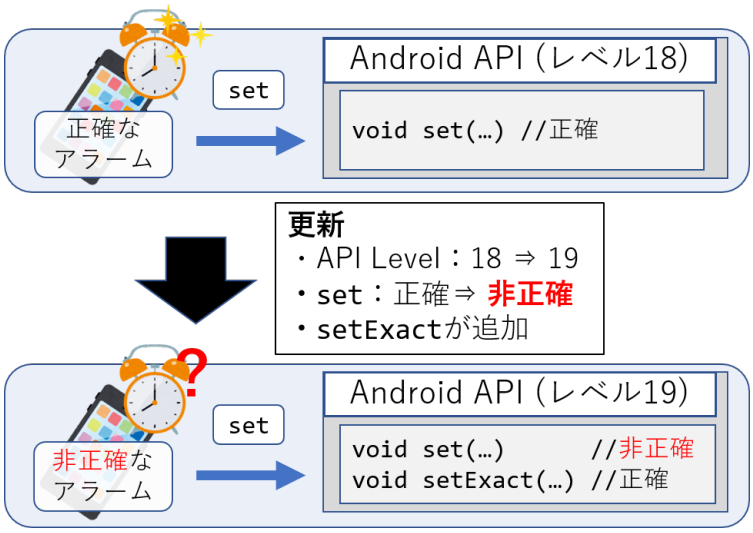

図 2.2 共有ライブラリのアップデートが原因でアラームの正確性が失われるアプリ

プリケーションの使用者が Android OS 6.0 へのアップデートを行うと、`AndroidHttpClient` に依存している一切の通信機能が稼働しなくなる。しかし、一方でこのアプリケーションを使用するために Android OS のアップデートを見送ると、このアプリケーションがアップデートのボトルネックとなり、http 通信関係以外のアップデートの恩恵を一切受けられなくなる。

また動作的な非互換性について、アプリケーションの動作に必要な関数・メソッドの実装が変更された場合、アプリケーションは必ずしも変更前のバージョンでの動作を保存するとは限らない。例えば `AlarmManager.set` を使用してアラーム機能を実装したアプリケーションがあるとする（図 2.2）。Android OS 4.4 へのアップデートに伴い、共有ライブラリの API レベルが 19 にアップデートされた状況を考える。前節で述べたように、このアップデートにより `AlarmManager` のスケジューリング戦略が変更される。このときアプリケーションの API レベル 19 への対応を行わないままでは、アプリケーションは依然として `AlarmManager.set` を使ったアラームである為、誤差を伴うアラームをトリガーする状態になる。仮にこのアプリケーションが正確なアラームを要件としていた場合、そのアプリケーションの誤作動の原因を Android のアップデートが作り出していることになってしまう。

これらの 2 つの具体例のように、あるプログラムを適切に機能させるために特定のバージョンのライブラリが必要な状況を、そのバージョンのライブラリへの依存性があるという。

2.2.2 第二の問題：バージョン違いのライブラリの共存不可

第二の問題は、同じ名前のライブラリのバージョン違いはアプリケーション中で共存できないという制約である。ライブラリを使用して制作されたアプリケーションは、ライブラリを頂点、ライブラリの依存性を辺とした依存性グラフを構成する。例えば、browserify[3] という Node.js のモジュールシステムをブラウザ上でも使用可能にする npm ライブラリ [4] の依存性グラフを可視化したもの [5] が図 2.3 である。また、アプリケーションから特定のライブラリまでの依存性を辿って構成される経路を依存性パスと呼ぶ。注目すべきなのは、依存性グラフ中のライブラリを表す各ノードには、ライブラリ名に"@"でバージョン番号が明記されていることである。browserify から inherit までの依存性パスを見ると、browserify のバージョン 16.5.0（browserify@16.5.0）はラ

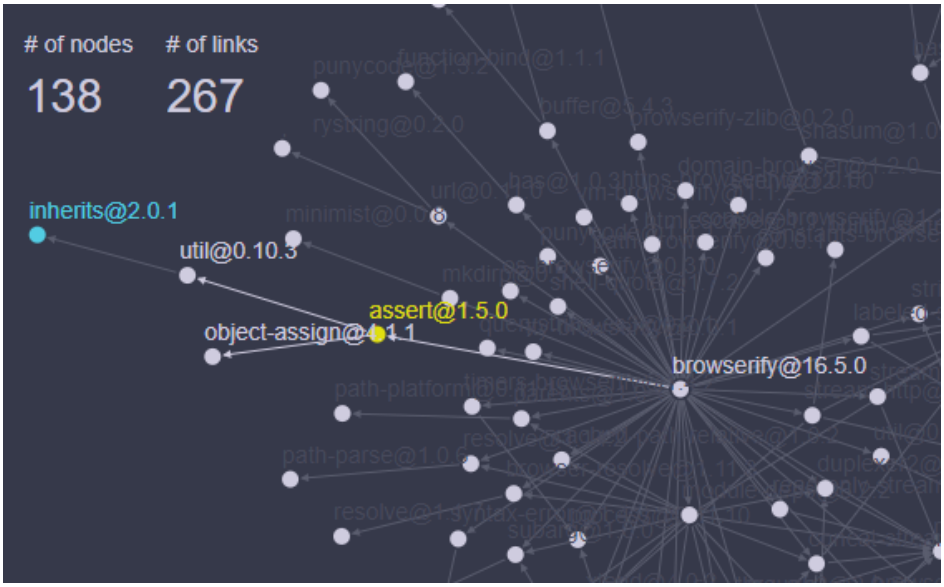

図 2.3　browserify@16.5.0 の依存性グラフ

イブラリ assert のバージョン 1.5.0(assert@1.5.0) に依存しており、さらに続けて assert@1.5.0 は util@0.10.3 に依存、最後に util@0.10.3 は inherit@2.0.1 に依存している。第一の問題で述べたように、バージョン違いによる非互換性はソフトウェアの誤作動の原因となる為、ライブラリの設計者は全ての依存ライブラリに自らの意図する実装を持つバージョンを指定する。browserify をはじめとするリリース済みの"正しい"依存性グラフを持つアプリケーションには、全ての依存ライブラリに唯一のバージョン番号が指定されている。

逆に二つの依存性パスのうち、両方に含まれるライブラリでバージョン違いが存在するものを、衝突する依存性パスと呼ぶ。衝突する依存性パスは第一の問題を発生させるため、"不正な"依存性グラフとみなされる。衝突する依存性パスは多くの場合、既存の依存性グラフを破壊的に更新するケースで発生する。以下の例は browserify で新しいライブラリ newLib の使用を検討するケースである (図 2.4)。newLib は util@0.11.5 に依存しており、Browsery から newLib への依存性パスを持ってしまうと衝突する依存性パスが発生する。このように、直接的に使用していないライブラリが原因で衝突する依存性パスが発生することもある。このときは newLib か assert のいずれかの使用を諦めなければならない。

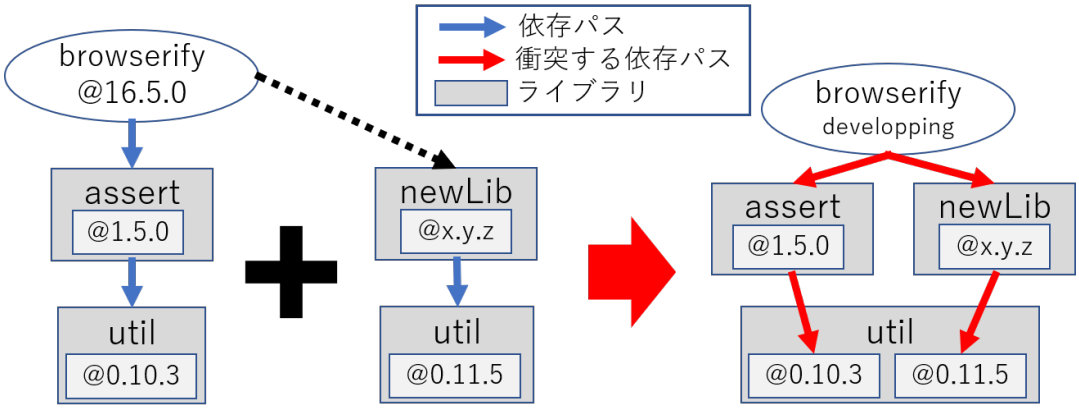

図 2.4　衝突する依存性パス

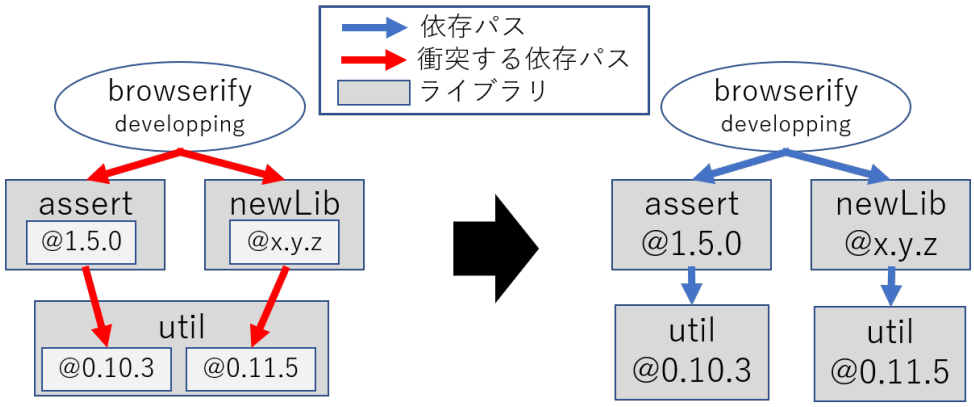

図 2.5 衝突する依存性パスの解消

2.3 衝突する依存性パスのナイーブな解消手法とその問題

2.3.1 ライブラリ名の変更

衝突する依存性パスの最も単純な解消手法は、ライブラリのバージョン違いの名前を変更することである。すなわち、ライブラリ Lib にバージョン 1 と 2 が存在する場合、ライブラリ Lib1 と Lib2 という個別のライブラリとして扱うということである。現代の標準的なプログラミング言語はいずれもライブラリを名前で区別するため、例えば関数 Lib.func のバージョン違いは、Lib1.func と Lib2.func というように、区別して参照することが可能となる。例えば前節で挙げた npm も各ライブラリの個別のバージョンを全て別のライブラリとして扱い、"不正な"依存性グラフを"正しい"依存性グラフに分割するという機能を備えている。例えば、図 2.4 における衝突する依存性パスは util の複数バージョンが原因で発生していたが、util@0.10.3 と util@0.11.5 を全く別のライブラリとすることで衝突は解消される（図 2.5）。

しかし静的型付け言語の立場に立つと、この手法は合理的な解決ではない。技術的な問題は、バージョン違いの 2 つのプログラムの型レベルの関係を全て失ってしまうことだ。以下に例を挙げる。

```haskell
module Lib1 where
  data MyList a = MyNil | MyCons a (MyList a) deriving Show
  myconcat :: MyList a -> MyList a -> MyList a
  myconcat MyNil ys         = ys
  myconcat (MyCons x xs) ys = MyCons x (myconcat xs ys)

module Lib2 where
  -- Lib1 と同じ定義
```

このプログラムは Lib という実装が同じ 2 つのバージョンを持つライブラリを、Haskell で書いたものである。ライブラリのバージョン違いは、先述の手法に従い Lib1 と Lib2 という 2 つの別の名前が与えられており、その中では同名のデータ型 MyList と関数 myconcat が定義されている。このとき以下のプログラムは ghc におけるコンパイルに失敗する。

```
import Lib1
import Lib2
main = do { print $
  Lib1.myconcat
    (Lib1.MyCons 1 Lib1.MyNil)
    (Lib2.MyCons 2 Lib2.MyNil) } -- この型は Lib2.MyList
{-
main.hs:6:6: error:
    • Couldn't match expected type  ‘Lib1.MyList Integer’
                  with actual type  ‘Lib2.MyList Integer’
      NB:  ‘Lib2.MyList’  is defined at Lib2.hs:2:3-64
           ‘Lib1.MyList’  is defined at Lib1.hs:2:3-64
-}
```

このプログラムは Lib1 と Lib2 それぞれに定義された MyList 型の値を、Lib1 に定義された myconcat で結合するプログラムである。Lib1.myconcat は引数として 2 つの Lib1.MyList 型の値を要求するが、上記のプログラムで第二引数の値は Lib2.MyList 型である。Lib1 と Lib2 の MyList の実装は同一のため、Lib のバージョン違いには完全な互換性がある。しかし、プログラマが Lib1.MyList = Lib2.MyList の等式を与えたくとも、ghc を初めとする現代のプログラミング言語の処理系にバージョン間の互換性の有無をキャプチャする仕組みは存在しない。この問題は、多くのバージョンを提供しており、プログラムの部分毎に互換性のあるバージョンの範囲が異なる実際のライブラリにおいてより複雑である。

2.3.2 モジュールファンクターの検討

モジュールファンクター [6] は高度なモジュールシステムを持つ ML 系言語 (Standard ML, OCaml 等) に見られる、ベースモジュールから新しいモジュールを生成する関数である。関数が引数でパラメータ化された関数であるのと同様に、モジュールファンクターは引数のモジュールがパラメータ化された関数であるとみなすことができる。モジュールファンクターを用いることで、ベースモジュールで定義された型レベルの関係を新しいモジュールに強制することが出来る。単純なアイデアとしては、バージョン間の互換性を型レベルで定義し、モジュールファンクターを用いてその関係を保存する新しいモジュールを生成できそうである。検討の為の例として、バージョン 2 はバージョン 1 への後方互換製を持つが、逆は成り立たないという関係を持った 2 つのバージョンのライブラリを表現する、以下の仮想 ML 言語で書かれたプログラムを考える。

```
module V1 = struct type t = T_V1 end
module V2 = struct type t = T_V2 end   (* T_V1 :> T_V2 *)
```

まず、各バージョンを表すベースモジュール V1 と V2 を定義する。ここで V1 と V2 の内部の型 t はそれぞれ T_V1 と T_V2 であり、後方互換製を表現する T_V1 :> T_V2 なる部分型関係があるとする。この部分型関係はリスコフの置換原則 [7]「型 T_V1 のデータを期待する文脈で型 T_V2 のデータを使用しても正しく動作する」を正しく満たしている。

次に各バージョンのライブラリを生成可能なファンクター MakeLib を定義する。

```
module type Version = sig type t ... end

module type MakeLib = sig
  module V : Version
  type tLib = V.t f
  val get : tLib -> ...
end

module MakeLib (V: Version) : (MakeLib with module V = V) =
struct
  module V = V
  type tLib = V.t f     (* f は共変な型コンストラクタ *)
  let get x = ...
end
```

ここで、ベースモジュールに定義された部分型関係を保存するように、MakeLib の内部型 tLib は共変な型コンストラクタ f で生成される。

ライブラリのユーザーのプログラムでは以下のようになる。

```
module Lib1 = MakeLib(V1)
module Lib2 = MakeLib(V2)

let join (x1 : Lib1.tLib) = Lib2.get (x1 :> Lib2.tLib) (* 型付け成功 *)
let join (x2 : Lib2.tLib) = Lib1.get (x2 :> Lib1.tLib) (* 型付け失敗 *)
```

ここまで定義したベースモジュールとモジュールファンクターを用いて、各バージョンのライブラリ Lib1, Lib2 を生成する。このとき、Lib1, Lib2 の内部型は共変の型コンストラクタにより Lib1.tLib :> Lib2.tLib の関係を持つ。従って 4 行目のプログラムは型付けに成功し、5 行目のプログラムは型付けに失敗する。

ここまでの検討で示したように、モジュールファンクターを用いた手法であればバージョン間の型の関係を保持することが可能である。一方で、この手法によるライブラリ設計には極めて強い制限がかかる。最大の問題はモジュールファンクターのインターフェースによって全バージョンのモジュール定義が決定され、更新時の自由なインターフェースの変更が不可能であることだ。またモジュール内の全てのプログラムにベースモジュールの部分型関係で定義された互換性を強要する為、例えばバージョン 2 とバージョン 3 で局所的に非互換な変更があった際に、大部分の互換性を保ったプログラムの互換性は捨てられてしまう。この例からわかるように、バージョン間の関係はモジュールレベルではなく、値レベルで定義可能な仕組みが合理的である。

第 3 章

λ_{VL} の概要

λ_{VL}[1] は依存性地獄の解決を目的としたプログラミング言語のコア計算である。λ_{VL} では、"バージョンの観点で安全"(バージョン安全) なプログラムを構成するために、構成要素のプログラムに要求するべきバージョンを解析することが可能である。λ_{VL} は型安全性の観点から依存性地獄にアプローチしており、バージョンの概念を単純型付きラムダ計算に導入した。λ_{VL} の型システムは、そのプログラムが利用可能なバージョンの集合を計算的リソースとして扱う。以下、λ_{VL} におけるコエフェクト計算の計算的リソースを版リソースと呼ぶ。

以下が λ_{VL} の特徴と、それを実現する λ_{VL} の言語要素の対応である。

- プログラムに複数のバージョンの定義を持たせることが可能
  - 第 3.1.1 節 Versioned Value
- プログラムの各バージョンの定義を区別して参照可能
  - 第 3.1.2 節 サスペンション演算子と抽出演算子
- プログラムの実行に必要なライブラリとそのバージョンを静的に解析可能
  - 第 3.1.3 節 型システムの概略

本章では第 3.1 節で λ_{VL} の諸概念の簡単な紹介をした後に、第 3.2 節で型・版多相性の拡張により表現可能となるプログラムを説明する。

3.1 λ_{VL} の諸概念

3.1.1 Versioned Value

λ_{VL} の特徴的な構文は **Versioned Value** である。Versioned Value は複数のバージョンで定義されたプログラムを一つにまとめ上げる構文であり、特定のバージョンに依存しないプログラムを書く機構を提供する。Versioned Value は構成要素としてバージョンタグとバージョン固有の定義のペアの集合、そしてその Versioned Value が評価されるべき文脈であるデフォルトバージョンを持つ。ここで Versioned Value に、あるバージョンに対応するバージョンタグが含まれる時、そのバージョンで使用可能であると呼ぶ。最も簡単な Versioned Value は以下のようなものである。

$$\{v18 :: 18,\ v19 :: 19 \mid v19\}$$

このプログラムは $v18$ と $v19$ で使用可能な Versioned Value であり、各バージョン固有の定義としてバージョンタグと同じ値 18 と 19 を持っている。この Versioned Value のデフォルトバージョンは $v19$ であり、これは次節

で述べる評価戦略に関係する。

バージョン違いで異なる定義を持つ関数も、Versioned Value として表現することが出来る。このような Versioned Value を特別に **Versioned Function** と呼ぶ。例えば以下の set は、第 2 章で取り上げた `AlarmManager.set` を λ_{VL} で書き直したプログラムである。

$$\text{set} = \{v18 :: \lambda x.(\text{非正確なアラームを設定}),\ v19 :: \lambda x.(\text{正確なアラームを設定}) \mid v19\}$$

第 2 章の例と同様、`AlarmManager.set` はレベル 18 と 19 でスケジューリング方式が異なる。これに従い、上記のプログラムで使用可能な $v18$ と $v19$ のバージョン固有の定義は、それぞれ非正確な時刻と正確な時刻を設定する関数となっている。

Versioend Function を Versioned Value に関数適用する際は **let** $[x] = t_1$ **in** t_2 構文を用いる。これは線形型システム [8, 9] 由来の構文であり、t_1 に与えられた計算的リソースを除去する方法を与える。例えば、以下の Unixtime を表現する Long 型の値を持った Versioned Value を myTime_v19 とする。

$$\text{myTime\_v19} = \{v19 :: 1577804400 \mid v19\}$$

さらに、先に定義された set を myTime_v19 に適用する。

$$\begin{aligned}&\textbf{let } [f] = \text{set } \textbf{in}\\ &\textbf{let } [t] = \text{myTime\_v19 } \textbf{in } f\ t\end{aligned}$$

このプログラムは以下のように評価される。はじめに set からデフォルトバージョンである $v19$ のバージョン固有の定義 (正確なアラームを設定する実装) を抽出し、その関数を f に束縛する。次に、同様に $v19$ のバージョン固有の定義を myTime_v19 から抽出し、その値を t に束縛する。最後に f を x に関数適用し、1577804400 の Unixtime の表現する 2020 年 0 時 0 分に正確にアラームを設定する。

Versioned Value の重要な使い方の一つとして、特定のバージョンに依存するプログラムの纏め上げがある。例として以下の mySetExact を挙げる。

$$\begin{aligned}\text{mySetExact} = &\textbf{let } [\text{set\_v18}] = \{\lambda x.(\text{ 正確な...}) \mid v18\} \textbf{ in}\\ &\textbf{let } [\text{setExact}] = \{\lambda x.(\text{ 正確な...}) \mid v19\} \textbf{ in}\\ &\{v18 :: \text{set\_v18},\ v19 :: \text{setExact}\}\end{aligned}$$

このプログラムは $v18$ と $v19$ の両方で正確なアラームを設定することを目的とした関数である。上記のプログラムにおいて、set_v18 と setExact はそれぞれ $v18$ と $v19$ のみにバージョン固有の定義が存在する、特定のバージョンへの依存性を持った変数である。最終行で set_v18 と setExact に対応するバージョンタグをつけ、新たに Versioned Value を生成している。このようにして定義された mySetExact は $v18$ と $v19$ で使用可能な Versioned Function であり、これを使用するプログラムは $v18$ や $v19$ といった一つだけのバージョンへの依存性を排除できる。

3.1.2 サスペンション演算子と抽出演算子

λ_{VL} では、Versioned Function と Versioned Value が共に複数のバージョンで使用可能な場合、それらの関数適用も複数のバージョンで計算が可能なプログラムとみなすことができる。これを実現するのがサスペンション演算子 $[*]$ と抽出演算子 $t.r$ である。ここで t は適当な Versioned Value で、r はバージョンタグである。

サスペンション演算子 [∗] は項を一つ受け取り、値が必要になるまで式の評価を一時停止する。例として、myTime を以下の様に定義する。

$$\text{myTime} = \{v18 :: 1577804400,\ v19 :: 1577804400 \mid v18\}$$

次に以下のように suspendedSet を与える。

$$\begin{aligned}\text{suspendedSet} = \ & \textbf{let}\ [f] = \text{set}\ \textbf{in} \\ & \textbf{let}\ [t] = \text{myTime}\ \textbf{in}\ [f\ t]\end{aligned}$$

このプログラムは $v18$ と $v19$ でそれぞれのスケジューリング方式でアラームを設定可能な、一時停止された計算を返す。

また抽出演算子 $t.r$ は、Versioned Value t からバージョンタグ r のバージョン固有の定義を抽出する。ここで t のデフォルトバージョンは、評価時に指定されたバージョンタグ r で上書きされる。例として、以下の suspendedSet_v19 を挙げる。

$$\begin{aligned}\text{suspendedSet\_v19} = \ & \textbf{let}\ [f] = \text{set}\ \textbf{in} \\ & \textbf{let}\ [t] = \text{myTime}\ \textbf{in}\ [f\ t].v19\end{aligned}$$

このプログラムは、先に構成した $v18$ と $v19$ の両方で計算が可能な一時停止されたプログラムから、$v19$ の計算（正確なアラームの設定）を取り出したものである。このとき myTime のデフォルトバージョン $v18$ は抽出演算子によって指定されたバージョンタグ $v19$ で上書きされ、全体としては $v19$ の計算を行う。

3.1.3 型システムの概略

特定のバージョンに依存しないプログラムの型

λ_{VL} の型システムは、特定のバージョンに依存しないプログラムに対し、そのプラグラムが使用可能なバージョンを型で明示することができる。特定のバージョンに依存しないプログラムのうち、最も簡単な例は先にも述べた以下の 2 つの Versioned Value である。

$$\begin{array}{ll}\text{myTime} : \square_{\{v18,v19\}}\mathsf{Int} & \text{set} : \square_{\{v18,v19\}}(\mathsf{Long} \to ()) \\ \text{myTime} = \{v18 :: 1577804400,\ v19 :: 1577804400 \mid v19\} & \text{set} = \{v18 :: \lambda x.(\text{非正確..}),\ v19 :: \lambda x.(\text{正確..}) \mid v19\}\end{array}$$

myTime と set に λ_{VL} が与える型は、それぞれ $\square_{\{v18,v19\}}\mathsf{Int}$ と $\square_{\{v18,v19\}}(\mathsf{Long} \to ())$ である。このように Versioned Value の型は、バージョン固有の定義の型（上記左の例では Int）に、使用可能なバージョンの集合 (同様に $\{v18, v19\}$) を"□"と共に添字付ける。myTime の型 $\square_{\{v18,v19\}}\mathsf{Int}$ は、このプログラムが $v18$ と $v19$ で使用可能な Int 型の値であることを示している。

より複雑な例は以下のような **let** $[x] = t_1$ **in** t_2 構文で構成されたプログラムの型である。

$$\begin{array}{ll}\text{suspendedSet} : \square_{\{v18,v19\}}() & \text{suspendedSet\_v19} : () \\ \begin{aligned}\text{suspendedSet} = \ & \textbf{let}\ [f] = \text{set}\ \textbf{in} \\ & \textbf{let}\ [t] = \text{myTime}\ \textbf{in}\ [f\ t]\end{aligned} & \begin{aligned}\text{suspendedSet\_v19} = \ & \textbf{let}\ [f] = \text{set}\ \textbf{in} \\ & \textbf{let}\ [t] = \text{myTime}\ \textbf{in}\ [f\ t].v19\end{aligned}\end{array}$$

これらの例のように、あるプログラムが λ_{VL} の特定のバージョンに依存しないプログラム同士の演算で与えられるとき、そのプログラムに与えられるバージョンタグ集合は**各 Versioned Value が持つバージョンタグ集合の**

共通部分である。suspendedSet の型につくバージョンタグ集合は、set と myTime の型の付与されたバージョンタグ集合の共通部分の $\{v18, v19\} = \{v18, v19\} \cap \{v18, v19\}$ である。注意すべきは suspendedSet_v19 の型は $\square_{\{v19\}}()$ ではなく、() であることだ。このように、抽出演算子はそのプログラムが持つ「どのバージョンで計算されるべきか」という文脈を取り外すことができる。

バージョン安全でないプログラムの型

λ_{VL} の型システムはバージョンの観点で安全でないプログラムをリジェクトできる。第一の ill-typed なプログラムは、構成要素のプログラムのバージョンタグ集合の共通部分を取ると空集合になるものである。例として、以下のように myTime_v20 と suspendedSet_v20 を定義する。

```
myTime_v20 : □{v20}Long                    suspendedSet_ill : (rejected)
myTime_v20 = {v20 : 1577804400 | v20}      suspendedSet_illt = let [f] = set in
                                                               let [t] = myTime_v20 in [f t]
```

これまでと同様に各構成要素のバージョンタグ集合の共通部分を取ると、空集合 $\emptyset = \{v18, v19\} \cap \{v20\}$ であるため、suspendedSet_ill は型付けに失敗する。この例からわかるように、λ_{VL} の型システムは少なくとも **1 つのバージョンタグにおいて、プログラム中のすべての Versioned Value で実装が存在する**ことを保証できる。これは、構造的な非互換性を含むプログラムにおいても、適当なバージョンを選択すれば必ず全ての実装が存在することに対応する。

第二の ill-typed なプログラムは、特定のバージョンへの依存性を持つ変数に、異なるバージョンタグをつけたものである。例として以下の mySetSwapped を挙げる。

```
mySetSwapped : (rejected)                                  mySetExact : □{v18,v19}(Long → ())
mySetSwapped = let [set_v18] = {λx.( 正確..) | v18} in                ...
               let [setExact] = {λx.( 正確..) | v19} in               ...
               {v18 :: setExact, v19 :: set_v18}                          {v18 :: set_v18, v19 :: setExact}
```

mySetSwapped は、mySetExact の最終行で、バージョンタグと適切なバージョン固有の定義を互い違いにしたものであるため、型付けに失敗する。上記二つの例からわかるように、λ_{VL} の型システムは**特定のバージョンへの依存性を持つプログラムが、不正なバージョンの文脈で計算されない**ことを保証できる。これは動作的な非互換性を含むプログラムにおいても、バージョンについて無矛盾な計算が可能であることに対応する。

3.2 型・版多相性の導入

本研究ではここまで紹介してきた λ_{VL} を、型多相性と版多相性によって拡張した。第 3.2.1 節では型多相性、第 3.2.2 節では版多相性について、本研究の拡張によりどのようなプログラムが表現可能となるかを述べる。

3.2.1 型多相性

本研究の最終目標は、現代的なモジュール機構を持つプログラミング言語にバージョンの概念を導入することである。本研究ではその為の第一歩として、λ_{VL} に**トップレベルでのみ全称型の型抽象を許したパラメータ多相**を導入する。モジュール機構を持つプログラミング言語は、パッケージ化されたモジュールは内部を隠蔽し、モ

ジュールを展開する際には複数の文脈でそれぞれ異なる型としてモジュールを表す必要がある。このモジュールの性質と密接に関係するのが多相性である。

現代のプログラミング言語の多相性にはいくつかの表現方法がある。そのうちの一つが全称型による型抽象であり、System F[10, 11] は全称型を導入した型システムの最初期の研究の一つだ。System F の多相性は、引数の型として多相型を持つことのできるパラメータ多相である。パラメータ多相では、具体的な型の代わりに型変数を用いて一つのプログラムに対して総称的な型付けを行うことができ、プログラムの必要な場所でそれを具体化することができる。System F は Haskell などのプログラミング言語の理論的基礎となっている。

一方で、存在型による多相性の実現も試みられてきた。Mitchell ら [12] は、型抽象は存在型として解釈できることを示した。抽象データ型やオブジェクトの表現を含むいくつかの研究があり、モジュール機構はその代表例である。初期の存在型ベースのモジュールシステムは展開されるたびに別の型が付与されるなどいくつかの問題を抱えていた [6]。そのため、近年は初期の欠点を克服した存在型モジュールの形式化も提案されている [13] が、OCaml など ML 方言のモジュールシステムは依存型ベースのもの [14, 15, 16] を採用している。また、初期の存在型ベースのモジュールシステムは全称型にエンコード可能なことが知られている [17] ため、本研究の拡張により λ_{VL} 上で単純なモジュール機構を実現することが可能となる。

型多相性による言語拡張は代数的データ型を含むプログラムの表現についても威力を発揮する。以下は第 2.3.1 節で挙げたプログラムである。

```
module Lib where
  data MyList a = MyNil | MyCons a (MyList a) deriving Show
```

`Lib.MyCons` の型は `forall a. a -> Lib.MyList a -> Lib.MyList a` であり、`a` でパラメータ化されている。このとき一つ目の引数の型と二つ目の引数の要素の型はどちらも `a` である。このように、型多相性の導入により、多相的な代数的データ型を表現することが可能になる。

3.2.2 版多相性

λ_{VL} の Versioned Value は、一つの特定のバージョンに依存しないプログラムを書くための言語機構を提供し、型システムにより不正なバージョンの組み合わせで計算が起こらないことを保証できた。一方で、より複雑なバージョン間のプログラムの関係性をキャプチャするためには、拡張前の λ_{VL} のような単純な言語機構では不十分である。

例えば、プログラム中に登場する計算的リソースに何等かの制約があるプログラムは拡張前の λ_{VL} で表現することはできない。最も単純な例は以下のプログラムである。このプログラムは、第 2.3.1 節で挙げたプログラムを、バージョンの概念を取り込んだモジュールシステムを持つ Haskell ライクな仮想言語で書き直したものだ。

```
module Lib[v1] where
  data MyList a = MyNil | MyCons a (MyList a) deriving Show
  myconcat :: MyList a -> MyList a -> MyList a
  myconcat MyNil ys         = ys
  myconcat (MyCons x xs) ys = MyCons x (myconcat xs ys)

module Lib[v2] where
  -- バージョン 2 の定義
```

プログラムの特定のバージョンへの依存性は、型に [*] で明示されている。例えば Lib[v1] は、$v1$ のモジュール Lib の意味である。今、Lib[v1] と Lib[v2] の両方に定義されている myconcat をバージョンに関して抽象化し、両方のバージョンの MyList を引数に取ることのできるプログラム myConcatPoly を定義したい。このようなプログラムの型を拡張前の λ_{VL} で表現することはできないが、版多相性を導入することで以下のように書ける。

```
1 import Lib
2
3 myConcatPoly : forall {r : Version}. MyList[r] a -> MyList[r] a -> MyList[r] a
4 myconcatPoly MyNil ys         = ys
5 myconcatPoly (MyCons x xs) ys = MyCons x (myconcat xs ys)
```

myConcatPoly の型はバージョンタグ r でパラメータ化されており、第一引数・第二引数・返り値の型は r で添え字付けられている。このプログラムの型により、第一引数と第二引数の MyList a 型の値にはともに同じバージョンで使用可能でなければならないという制約があることを型レベルで保証することが可能となる。

第 2 章で述べたように、現実のソフトウェア開発で出現するバージョン違いのプログラムの関係性はより複雑である。版多相性の導入により型レベルにバージョン変数を含めることが可能になり、これはバージョンに関する制約を型検査で行う手法への洞察を与える。型レベルでのバージョンに関する制約については第 6.2 節で論じる。

第 4 章

拡張 λ_{VL}

本研究では、オリジナルの λ_{VL} にカインドベースの拡張により型多相性と版多相性を与えた。オリジナルの λ_{VL} は PCF[18] をベースにしたコエフェクト計算である $\ell\mathcal{R}$PCF[19] に基づき、バージョンの概念を単純型付きラムダ計算に導入した計算体系である。コエフェクト計算への多相性を導入する手法は、λ_{VL} と同じく $\ell\mathcal{R}$PCF をベースとしたコエフェクト計算をコア計算に持つプログラミング言語 Granule[20] に従う。

4.1 構文

λ_{VL} の構文は以下のように与えられる。なお、λ_{VL} における計算的リソースは版である。

$$
\begin{array}{rrcl}
\text{(項)} & t & ::= & \underbrace{x \mid t_1\ t_2 \mid \lambda p.t}_{\lambda\text{-terms}} \mid \underbrace{n}_{\text{integers}} \mid \underbrace{[t]}_{\text{box}} \mid \underbrace{\{\overline{l = t} \mid l_i\} \mid t.l \mid \langle \overline{l = t} \mid l_i \rangle}_{\text{versioned term}} \\
\text{(ラベル)} & l & ::= & \underbrace{l_1, l_2, \cdots}_{\text{labels}} \\
\text{(パターン)} & p & ::= & \underbrace{x}_{\text{variables}} \mid \underbrace{\_}_{\text{wildcard}} \mid \underbrace{[x]}_{\text{unbox}}
\end{array}
$$

λ_{VL} の構文は、通常のラムダ計算の構文全て (変数 x, 関数適用 $t_1\ t_2$, ラムダ抽象 $\lambda p.t$) に加え、コンストラクターとして n、リフト $[t]$、Versioned Value $\{\overline{l = t} \mid l_i\}$、抽出演算 $t.l$、中間項 $\langle \overline{l = t} \mid l_i \rangle$ である。n は Int 型の値のコンストラクターである。また $[t]$ は版リソースの導入規則であり、バージョン無し計算のバージョン有り計算へのリフトを可能とする構文である。なお、$[t]$ は第 2 章で述べた！コンストラクタ (サスペンド演算子) と同一のものである。Versioned Value は第 3 章の説明の通り各バージョン固有の定義をバージョンタグと共にパッケージ化する構文で、デフォルトバージョン l_i はその Versioned Value 中に存在するバージョンタグの中に存在する。抽出演算 $t.l$ は Versioned Value からバージョン固有の定義を一つ取り出す演算である。また、中間項 $\langle \overline{l = t} \mid l_i \rangle$ はデフォルトバージョンでの評価が遅延される項を表現している。これは Versioned Value の評価の途中にのみ出現し、ユーザーのプログラムに存在しないことを想定している。

また、これらの構文に対応するパターンとして、変数とワイルドカード、整数、アンボックスパターンが存在している。最後に、第 3 章で登場した項に与えられたバージョンを除去する clet 構文は、以下のように $[p]$ を引数に取る λ 抽象の構文糖衣として与えられる。

$$\textbf{let}\ [p] = t_1\ \textbf{in}\ t_2 \triangleq (\lambda[p].t_2)\ t_1$$

λ_{VL} の型とカインドは以下のように与えられる。

(型)	A, B	::=	$A \to B \mid K \mid \alpha \mid A\ B \mid \Box_c A$
(カインド)	κ	::=	$\mathsf{Type} \mid \mathsf{Coeff} \mid \kappa_1 \to \kappa_2 \mid \uparrow A$
(型演算子)	op	::=	$\otimes \mid \oplus$
(型コンストラクタ)	K	::=	$\mathsf{Int} \mid \mathsf{Version}$
(型スキーム)	T	::=	$\forall \{\overrightarrow{\alpha : \kappa}\}.A$

型は関数型 $A \to B$、型構築子 K、型変数 α、型適用 $A\,B$、二項演算子 $A \,\mathrm{op}\, B$、版付き型 $\Box_c A$ からなる。ここで c は第 4.2 節で定義される版リソースである。λ_{VL} の全ての型は暗黙に線形性を要求しており、全ての項変数は一回のみ使用される。伝統的に線形性を要求する体系の関数型は $\multimap$ で表現することが多いが、本論文では簡単のために通常の矢印 $\to$ を用いる。カインドは型カインド Type、コエフェクトカインド Coeff、高階カインド型の為の関数空間 $\kappa_1 \to \kappa_2$、型 A をカインドにリフトする $\uparrow A$ で構成される。

型コンストラクタ K は組み込みの型で構成される。Int 型はカインド Type に分類され、Version 型はカインド Coeff、残りの 2 つは高階カインド Coeff → Coeff である。

先にも述べた通り、λ_{VL} はトップレベルの型宣言でのみ、型スキームを通して多相性を導入することが出来る。ここで $\overrightarrow{\alpha : \kappa}$ はカンマで区切られた (型変数):(カインド) の列である。最後に、トップレベル定義は以下のように型スキームとパターンに率いられる等式の列である。

$$Def ::= x : T; \overrightarrow{x\ p_{i1} \ldots p_{in} = t_i} \qquad \text{(トップレベル定義)}$$

4.2 版リソース代数

λ_{VL} の Versioned Value には版リソースを用いた型付けが行われる。ある項 t に $t : \Box_c A$ という型付けがなされたとき、型システム上では t は版リソース c を要求する型 A であることを意味する。このような型をバージョン付き型と呼ぶ。ここで版リソース r は版リソース代数 $(R_{\mathsf{ver}}, \oplus, 0, \otimes, 1, \leq)$ によって特徴づけられるコエフェクトである。R_{ver} を用いて、λ_{VL} に登場する版リソースは以下のように定義される。

(版リソース)	r	::=	$\alpha \mid 0 \mid 1 \mid r_1 \oplus r_2 \mid r_1 \otimes r_2$

最初の構文は版リソース変数 α である。この構文は版リソースをパラメタ化し、λ_{VL} で版多相性を扱うことを可能にする。残る構文は以下に定義される版リソース代数上の演算・順序により与えられる $0, 1, \oplus, \otimes, \leq$ で定義されるものである。

以下に版リソース代数の定義を与える。多くのコエフェクト計算と同様に、版リソース代数は半順序を持った半環 (構造的半環) の上に定義される。

定義 4.2.1 [構造的半環]

$(\mathcal{R}, \oplus, 0, \otimes, 1, \leq)$ が構造的半環であるとは、$\mathcal{R}$ が加法・乗法・半順序を持ち、以下の性質を満たすことである。

- $(\mathcal{R}, \oplus, 0)$ は加法 $\oplus$ について単位元 0 を持つ可換モノイドを成す。すなわち以下の 1, 2, 3 を満たす。
  1. 結合則：$\forall a, b, c \in \mathcal{R}.\ (a \oplus b) \oplus c = a \oplus (b \oplus c)$
  2. 加法単位元の存在：$\forall a \in \mathcal{R}.\ 0 \oplus a = a \oplus 0 = a$
  3. 交換則：$\forall a, b \in \mathcal{R}.\ a \oplus b = b \oplus a$
- $(\mathcal{R}, \otimes, 1)$ は乗法 $\otimes$ について単位元 1 を持つモノイドを成す。すなわち以下の 1, 2 を満たす。

1. 結合則：$\forall a, b, c \in \mathcal{R}.\ (a \otimes b) \otimes c = a \otimes (b \otimes c)$
2. 乗法単位元の存在：$\forall a \in \mathcal{R}.\ 1 \otimes a = a \otimes 1 = a$

- 加法 $\oplus$ と乗法 $\otimes$ は分配的である。すなわち以下の 1, 2 を満たす。
  1. $\forall a, b, c \in \mathcal{R}.\ a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$
  2. $\forall a, b, c \in \mathcal{R}.\ (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- 加法単位元 0 は乗法において $\mathcal{R}$ を零化する。
  1. $\forall a \in \mathcal{R}.\ a \otimes 0 = 0 \otimes a = 0$
- $\leq$ は半順序である。即ち以下の 1, 2, 3 を満たす。
  1. 反射律：$\forall a \in \mathcal{R}.\ a \leq a$
  2. 推移律：$\forall a, b, c \in \mathcal{R}. a \leq b \land b \leq c \Rightarrow a \leq c$
  3. 反対称律：$\forall a, b \in \mathcal{R}. a \leq b \land b \leq a \Rightarrow a = a$

以下に定義されるように、版リソース代数は構造的半環である。以後型導出規則などで一般の版リソース代数についての定義であることを強調したいとき、文字色を青にする。

定義 4.2.2 [版リソース代数と Version コエフェクト]

コエフェクト型 Version は要素としてバージョンタグを持つ構造的半環 R_{ver} で特徴づけられる。R_{ver} の加法 $\oplus$・乗法 $\otimes$ および加法単位元 0・乗法単位元 1 と半順序 $\leq$ は以下のように定義する。ここで $\subseteq$ は通常の部分集合関係である。

$$0 = \bot \quad 1 = \emptyset \qquad \bot \leq \mathsf{r} \qquad \frac{\mathsf{r}_1 \subseteq \mathsf{r}_2}{\mathsf{r}_1 \leq \mathsf{r}_2}$$

$$r_1 \oplus r_2 = \begin{cases} r_1 & r_2 = \bot \\ r_2 & r_1 = \bot \\ r_1 \cup r_2 & \text{otherwise} \end{cases} \qquad r_1 \otimes r_2 = \begin{cases} \bot & (r_1 = \bot) \lor (r_2 = \bot) \\ r_1 \cup r_2 & \text{otherwise} \end{cases}$$

4.3 型システム

λ_{VL} の型導出, カインド導出は以下の形式で与えられる。

(型導出)	$D; \Sigma; \Gamma \vdash t : A$
(カインド導出)	$\Sigma \vdash \alpha : \kappa$

ここで D, Σ, Γ はそれぞれトップレベル定義の環境, 型変数の環境, 項変数の環境を表すメタ変数である。各環境は以下のように定義される。

(トップレベル定義環境)	D	::=	$\emptyset \mid D, x : T$
(型変数環境)	Σ	::=	$\emptyset \mid \Sigma, \alpha : \kappa, \alpha :_{\exists} \kappa$
(項変数環境)	Γ, Δ	::=	$\emptyset \mid \Gamma, x : A \mid \Gamma, x : [A]_r$

トップレベル定義環境は、型スキーム $C : T$ の列である。型変数環境は 2 種類のカインド仮定の列である。ここで、型の単一化に用いられる仮定には $\exists$ を付与し、区別している。項変数環境は拡張前の λ_{VL} の項変数環境と同様に定義されており、拡張前の λ_{VL} の型導出は拡張 λ_{VL} の型判断を $\emptyset; \emptyset; \Gamma \vdash t : A$ と特殊化したものである。

λ_{VL} のカインド導出規則 $\boxed{\Sigma \vdash A : \kappa}$

$$\frac{\Sigma \vdash A : \mathsf{Type} \quad \Sigma \vdash B : \mathsf{Type}}{\Sigma \vdash A \to B}\ (\kappa_{\to}) \qquad \frac{\Sigma \vdash A : \kappa_1 \to \kappa_2 \quad \Sigma \vdash B : \kappa_1}{\Sigma \vdash A\, B : \kappa_2}\ (\kappa\mathrm{APP})$$

$$\frac{}{\Sigma, \alpha : \kappa \vdash \alpha : \kappa}\ (\kappa\mathrm{VAR}) \qquad \frac{}{\Sigma, \alpha :_{\exists} \kappa \vdash \alpha : \kappa}\ (\kappa\exists\mathrm{VAR}) \qquad \frac{A \in \{\mathsf{Int}\}}{\Sigma \vdash A : \mathsf{Type}}\ (\kappa\mathrm{TYPES})$$

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \quad \Sigma \vdash r : \uparrow R \quad \Sigma \vdash A : \mathsf{Type}}{\Sigma \vdash \square_r A : \mathsf{Type}}\ (\kappa\square) \qquad \frac{\mathrm{op} \in \{\oplus, \otimes\} \quad \Sigma \vdash R : \mathsf{Coeff} \quad \Sigma \vdash A : \uparrow R \quad \Sigma \vdash B : \uparrow R}{\Sigma \vdash A\, \mathrm{op}\, B : \uparrow R}\ (\kappa\mathrm{OP})$$

$$\frac{\Sigma \vdash R : \mathsf{Coeff}}{\Sigma \vdash 0 : \uparrow R}\ (\kappa 0) \qquad \frac{\Sigma \vdash R : \mathsf{Coeff}}{\Sigma \vdash 1 : \uparrow R}\ (\kappa 1) \qquad \frac{\Sigma \vdash R : \mathsf{Coeff} \quad l : \mathsf{Label}}{\Sigma \vdash \{l\} : \uparrow R}\ (\kappa\mathrm{LABEL}) \qquad \frac{K \in \{\mathsf{Version}\}}{\Sigma \vdash K : \mathsf{Coeff}}\ (\kappa\mathrm{Co})$$

図 4.1 λ_{VL} のカインド導出規則

まずはじめに第 4.3.1 節で拡張 λ_{VL} のカインド導出について説明する。次に第 4.3.3 節でパターンの型付けについて述べ、最後に第 4.3.4 節で各構文の型付け規則を説明する。

4.3.1 カインド導出

図 4.1 は λ_{VL} の型 A の型変数環境 Σ におけるカインド κ の導出 $\Sigma \vdash A : \kappa$ を定める規則群である。初めの 2 つの規則 (κVAR) と ($\kappa_{\to}$) はカインドベースの型システムに標準的に見られる規則であり、それぞれ関数型のカインドと関数適用の型のカインドを導出する。型変数のカインドは (κVAR) と ($\kappa\exists$VAR) のいずれかで導出される。同様に具体型は (κTYPES) で導出される。先述のように本論文の体系では Int 型のみが組み込みの具体型であり、(κTYPES) の前提に示されている。

バージョン付き型は ($\kappa\square$) で導出される。続く (κOP) は版リソース代数上の演算を型レベル演算として導出する規則である。規則の前提にある $\oplus$ と $\otimes$ は各コエフェクトの和積演算で具体化される。次の 3 つの規則 (κ0), (κ1), (κlabel) は版リソース代数の元のカインドを、それぞれ 0, 1, 単一の何らかのバージョンタグ（ラベル）として導出する。先に述べた (κOP) と合わせ、全てのバージョンは 0, 1, 単一の何らかのバージョンタグと $\oplus, \otimes$ の演算結果として導出可能でなければならない。最後に、残りの 2 つの規則 (κCo) は組み込みの型コンストラクター K のカインドを導出する。先述したように λ_{VL} の組み込みの単階コエフェクトは Version のみであり、(κCo) の前提に示されている。

4.3.2 型代入と型単一化

型・版多相性の導入に伴い、後述の型導出規則に型代入 θ を使用している。個々の型代入は型変数 α、具体型 A を用いて $\alpha \mapsto A$ で表される。型代入は型・カインド・版リソース・環境 Γ・型変数環境 Σ・型代入に適用でき、具体型への適用は $\theta\, A$ で表す。

図 4.2 は λ_{VL} の型単一化導出 $\Sigma \vdash A \sim A \triangleright \theta$ の導出規則である。このように型単一化は型変数環境 Σ の下での

λ_{VL} の型単一化の導出規則 $\boxed{\Sigma \vdash A \sim A \rhd \theta}$

$$\frac{\Sigma \vdash A' \sim A \rhd \theta_1 \quad \Sigma \vdash \theta_1 B \sim \theta_1 B' \rhd \theta_2}{\Sigma \vdash A \to B \sim A' \to B' \rhd \theta_1 \uplus \theta_2}\,(U_{\to}) \qquad \frac{\Sigma \vdash A \sim A' \rhd \theta_1 \quad \Sigma \vdash \theta_1 B \sim \theta_1 B' \rhd \theta_2}{\Sigma \vdash A\,B \sim A'\,B' \rhd \theta_1 \uplus \theta_2}\,(U_{\mathrm{APP}})$$

$$\frac{(\alpha : \kappa) \in \Sigma}{\Sigma \vdash \alpha \sim \alpha \rhd \emptyset}\,(U_{\mathrm{VAR}=}) \qquad \frac{\Sigma \vdash A : \kappa \quad \Sigma \vdash (\alpha :_{\exists} \kappa) \in \Sigma}{\Sigma \vdash \alpha \sim A \rhd \alpha \mapsto A}\,(U_{\mathrm{VAR}\exists}) \qquad \frac{}{\Sigma \vdash K \sim K \rhd \emptyset}\,(U_{\mathrm{CON}})$$

$$\frac{\Sigma \vdash A : \kappa}{\Sigma \vdash A \sim A \rhd \emptyset}\,(U_{=}) \qquad \frac{\Sigma \vdash A \sim A' \rhd \theta_1 \quad \Sigma \vdash \theta_1 c \sim \theta_1 c' \rhd \theta_2}{\Sigma \vdash \Box_c A \sim \Box_{c'} A' \rhd \theta_1 \uplus \theta_2}\,(U_{\Box})$$

図 4.2 λ_{VL} の型単一化の導出規則

型の合同関係 $A \sim A$ であり、型代入 θ を生成する。$U_{\to}$ と $U_{\mathrm{APP}}, U_{\Box}$ のように前提が複数ある規則では、前の前提を単一化することで生成された型代入は後ろの代入に適用され、その上で型単一化を行う。詳細な型代入の定義については付録 A.1 を見よ。

また、型代入の合成 $\theta_1 \uplus \theta_2$ は以下のように定義される。

定義 4.3.1 [型代入の合成 $\uplus$]

型変数環境 Σ と二つの well-formed な型代入 θ_1 と θ_2 について、$\theta_1 \uplus \theta_2$ を以下のように定義する。

$$\begin{aligned} \emptyset \uplus \theta_2 &= \theta_2 \\ (\theta_1, \alpha \mapsto A) \uplus \theta_2 &= \begin{cases} (\theta_1 \uplus (\theta_2 \backslash \alpha) \uplus \theta), \alpha \mapsto \theta A & \theta_2(\alpha) = B \wedge \Sigma \vdash A \sim B \rhd \theta \\ (\theta_1 \uplus \theta_2), \alpha \mapsto A & \alpha \notin \mathrm{dom}(\theta_2) \end{cases} \end{aligned}$$

この計算は型変数 α が θ_2 に含まれるときは、その θ_2 における適用後の型 $\theta_2(\alpha) = B$ と θ_1 における適用後の型 A から生成される新たな型代入 θ を計算し、それを用いて新しい型代入 $\alpha \mapsto \theta A$ を生成する。θ_1 と θ_2 の両方に α が含まれる場合、新しい型代入 $\Sigma \vdash A \sim B \rhd \theta$ の計算は失敗しうることに注意が必要である。

$U_{\Box}$ に示されるように、型代入は版リソースにも行う。これは版多相性により導入された版変数は版リソースにも含まれるためである。最後に、全てのパラメータ化された型変数は $U_{\mathrm{VAR}=}, U_{\mathrm{VAR}\exists}$ と $U_{=}$ のいずれかによって単一化される。この際、具体型にマッチした型変数 α は型変数環境に $\alpha :_{\exists} \kappa$ のカインドを要求し、型代入 $\alpha \mapsto A$ を生成する。最後に、型代入の安全性について述べる。

定義 4.3.2 [型代入の安全性]

型変数環境 Σ と型代入 θ が与えられたとき、θ が安全な型代入であるとは以下の規則で導出可能なことを言う。

$$\frac{}{\Sigma \vdash \emptyset}\,(\textsc{empty}) \qquad \frac{\Sigma \vdash \theta \quad (\alpha : \kappa) \in \Sigma \quad \Sigma \backslash \{\alpha : \kappa\} \vdash A : \kappa}{\Sigma \vdash \theta \uplus (\alpha \mapsto A)}\,(\textsc{ext})$$

安全な型代入はカインドを変更しない。　また、代入された型 A の well-kindness の導出の前提には、α の取り除かれた型変数環境での well-kindness の導出が必要である。

λ_{VL} のパターン型導出 $\boxed{D; \Sigma; r :?R \vdash p : A \rhd \Gamma}$

$$\frac{\Sigma \vdash A : \mathsf{Type}}{D; \Sigma; - \vdash x : A \rhd x : A}\ (\textsc{pVar}) \qquad \frac{\Sigma \vdash A : \mathsf{Type}}{D; \Sigma; r : R \vdash x : A \rhd x : [A]_{r:R}}\ ([\textsc{pVar}])$$

$$\frac{0 \sqsubseteq r \quad D; \Sigma \vdash A : \mathsf{Type}}{D; \Sigma; r : R \vdash \_ : A \rhd \emptyset}\ (\textsc{p\_}) \qquad \frac{D; \Sigma; r : R \vdash x : A \rhd \Delta \quad \Sigma \vdash r : \uparrow R}{D; \Sigma; - \vdash [x] : \Box_r A \rhd \Delta}\ (\textsc{p}\Box)$$

λ_{VL} のトップレベル項等式 $\boxed{D \vdash Eqn : T}$

$$\frac{D; \overrightarrow{\alpha : \kappa}; - \vdash p_i : B_i \rhd \Delta_i \quad D; \overrightarrow{\alpha : \kappa}; \Delta_0, \ldots, \Delta_n \vdash t : A}{D \vdash x\ p_1\ ..\ p_n = t : \forall \{\overrightarrow{\alpha : \kappa}\}.(B_0 \to \ldots \to B_n \to A)}\ (\textsc{Eqn})$$

図 4.3 λ_{VL} のパターン型導出規則

4.3.3 パターンマッチ

λ_{VL} のパターンマッチは、他の関数型言語 (例えば ML[21]) のパターンマッチ機構と同様、値をパターンとみなして適切な変数束縛を与える。他の言語と異なるのは、パターンが版リソースの消費を引き起こし、それをキャプチャすることが可能な点である。これは第 4 章冒頭で述べた **let** $[p] = t_1$ **in** t_2 の糖衣構文としてのラムダ抽象の機能に当たる。この手法はパターンマッチを持つコエフェクト計算が可能なプログラミング言語 Granule[20] に従う。

λ_{VL} のパターン型導出は $D; \Sigma; r :?R \vdash p : A \rhd \Gamma$ で与えられる。パターン型導出はトップレベル定義環境 D と型変数環境 Σ が与えられたとき、パターン p に型 A を与え、新たな項変数環境 Δ を生成する。この型導出には新たな追加の環境 $r :?R$ が与えられる。

$$r :?R ::= - \mid r : R \qquad \text{(版リソース環境)}$$

これは版リソースをパターンマッチで消費したとき、版リソースの情報を保持しておくための環境である。これによって保持された版リソース情報は、[pVar] で生成された項変数環境に与えられる。

注意すべきなのは、λ_{VL} では以下の [p□] 規則が定義されていないことである。

$$\frac{D; \Sigma; s : S \vdash p : A \rhd \Delta \quad \Sigma \vdash r' : \uparrow R' \quad \mathrm{flatten}(r, R, r', R') = (s, S)}{D; \Sigma; r : R \vdash [p] : \Box_{r'} A \rhd \Delta}\ ([\textsc{p}\Box])$$

これは p□ 規則における、二重の版リソースの除去を認める為の規則である。Granule では多種多様なリソース代数に対して二重のリソース除去をした際の演算 (flatten) を定めており、それら全てに対してリソースの二重除去が可能となっている。現段階の λ_{VL} ではパターンマッチによる二重の版リソース除去を許していない。これは版リソースを二重に除去した際に、[pVar] で項変数に与えるべき版リソースの解釈が自明でない為である。この定義・解釈を考案することは今後の課題である。

λ_{VL} の型導出規則 $\boxed{D;\Sigma;\Gamma \vdash t : A}$

$$\frac{\Sigma \vdash A : \mathsf{Type}}{D;\Sigma;x : A \vdash x : A}\ (\textsc{var}) \qquad \frac{D;\Sigma;- \vdash p : A \rhd \Delta \quad D;\Sigma;\Gamma,\Delta \vdash t : B}{D;\Sigma;\Gamma \vdash \lambda p.t : A \to B}\ (\textsc{abs}) \qquad \frac{D;\Sigma;\Gamma_1 \vdash t_1 : A \to B \quad D;\Sigma;\Gamma_2 \vdash t_2 : A}{D;\Sigma;\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B}\ (\textsc{app})$$

$$\frac{(x : \forall\{\overrightarrow{\alpha : \kappa}\}.A) \in D \quad \theta, \Sigma' = \mathsf{inst}(\overrightarrow{\alpha : \kappa})}{D;\Sigma,\Sigma';\emptyset \vdash x : \theta A}\ (\mathrm{C}) \qquad \frac{}{D;\Delta;\emptyset \vdash n : \mathsf{Int}}\ (\textsc{int})$$

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \quad D;\Sigma;\Gamma \vdash t : A}{D;\Sigma;\Gamma + [\Delta]_{0:R} \vdash t : A}\ (\textsc{weak}) \qquad \frac{\Sigma \vdash R : \mathsf{Coeff} \quad D;\Sigma;\Gamma,x : A \vdash t : B}{D;\Sigma;\Gamma,x : [A]_{1:R} \vdash t : B}\ (\textsc{der}) \qquad \frac{\Sigma \vdash r : \uparrow R \quad \Sigma \vdash R : \mathsf{Coeff} \quad D;\Sigma;[\Gamma] \vdash t : A}{D;\Sigma;r \cdot [\Gamma] \vdash [t] : \Box_r A}\ (\textsc{pr})$$

$$\frac{D;\Sigma;[\Gamma_i] \vdash t_i : A}{D;\Sigma;\bigcup_i(\{l_i\} \cdot [\Gamma_i]) \vdash \{\overline{l = t} | l_i\} : \Box_{\overline{l}} A}\ (\textsc{ver}) \qquad \frac{D;\Sigma;[\Gamma_i] \vdash t_i : A}{D;\Sigma;\bigcup_i(\{l_i\} \cdot [\Gamma_i]) \vdash \langle\overline{l = t} | l_i\rangle : A}\ (\textsc{veri})$$

$$\frac{D;\Sigma;\Gamma \vdash t : \Box_r A \quad l \in r}{D;\Sigma;\Gamma \vdash t.l : A}\ (\textsc{rstr}) \qquad \frac{D;\Sigma;\Gamma,x : [A]_r,\Gamma' \vdash t : B \quad r \sqsubseteq s}{D;\Sigma;\Gamma,x : [A]_s,\Gamma' \vdash t : B}\ (\sqsubseteq)$$

4.3.4 型導出

ここまでの全ての導出規則を用いて、λ_{VL} の型導出規則を定義する。λ_{VL} の型付け規則は、トップレベル定義環境 D, 型変数環境 Σ, 項変数環境 Γ における項 t の型 A を導出する。各型付け規則を順番に説明する。

$$\frac{\Sigma \vdash A : \mathsf{Type}}{D;\Sigma;x : A \vdash x : A}\ (\textsc{var}) \qquad \frac{D;\Sigma;- \vdash p : A \rhd \Delta \quad D;\Sigma;\Gamma,\Delta \vdash t : \theta B}{D;\Sigma;\Gamma \vdash \lambda p.t : A \to B}\ (\textsc{abs}) \qquad \frac{D;\Sigma;\Gamma_1 \vdash t_1 : A \to B \quad D;\Sigma;\Gamma_2 \vdash t_2 : A}{D;\Sigma;\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B}\ (\textsc{app})$$

(VAR)・(ABS)・(APP) の定義は通常のラムダ計算の型付け規則と同様であるが、トップレベル定義環境と型変数環境が追加されている。(VAR) では型 A が well-defined なものかどうかをカインド型付けで検査する。(ABS) は本研究により拡張され、項変数抽象はパターンマッチとなった。パターン p の導出によって生成された項変数環境をラムダ抽象の本体の部分の型付けで用いる。注意すべきなのは、(APP) の結論の項変数環境 $\Gamma_1 + \Gamma_2$ や、(VER),(VERI) に出現する $\bigcup_i \Gamma_i$ である。この環境の結合 + は以下のように定義される。

定義 4.3.3 [項変数環境の結合 + & $\bigcup$]

交わりの無い 2 つの項変数環境は","で結合可能である。さらに両方の項変数環境に現れるコエフェクト付きの仮定は、構造的半環の加算 $\oplus$ を用いて結合可能である。このとき、2 つの項変数環境の結合 + を以下のように定義する。

$$(\Gamma, x : A) + \Gamma' = (\Gamma + \Gamma'), x : A \quad \text{iff } x \notin |\Gamma'| \qquad \emptyset + \Gamma = \Gamma$$
$$\Gamma + (\Gamma', x : A) = (\Gamma + \Gamma'), x : A \quad \text{iff } x \notin |\Gamma| \qquad \emptyset + \Gamma = \Gamma$$
$$(\Gamma.x : [A]_r) + (\Gamma', x : [A]_s) = (\Gamma + \Gamma'), x : [A]_{(r \oplus s)}$$

また、以上で定義した + を用い、項変数環境の総和を以下のように定める。

$$\bigcup_{i \in n} \Gamma_i = \Gamma_1 + \cdots + \Gamma_n$$

$$\frac{(x : \forall\{\overrightarrow{\alpha : \kappa}\}.A) \in D \quad \theta, \Sigma' = \mathsf{inst}(\overrightarrow{\alpha : \kappa})}{D; \Sigma, \Sigma'; \emptyset \vdash x : \theta A}\ (\mathrm{C}) \qquad \frac{}{D; \Delta; \emptyset \vdash n : \mathsf{Int}}\ (\textsc{int})$$

次に、(C) はトップレベル定義を環境から取り出すための規則である。$\mathsf{inst}(\overrightarrow{\alpha : \kappa})$ により、量化された型変数（コエフェクト変数含む）が具体化され、各 α の名前を付け替えたフレッシュな型変数の環境 Σ' と、量化された型変数を具体型に射影する置換 θ が与えられる。inst を以下のように定義する。

定義 4.3.4 [型具体化]
型スキーム $\forall\{\overrightarrow{\alpha : \kappa}\}.A$ が与えられたとき、型具体化 $\mathsf{inst}(\overrightarrow{\alpha : \kappa}) = \theta, \Sigma$ を以下の様に定義する。

$$\mathsf{inst}(\emptyset) = \emptyset, \emptyset$$
$$\mathsf{inst}((\alpha_0 : \kappa_0), \overrightarrow{\alpha : \kappa}) = (\alpha_0 \mapsto \alpha_0') \cup \theta_{\mathsf{inst}(\overrightarrow{\alpha:\kappa})}, \{\alpha_0' : \kappa\} \cup \Sigma_{\mathsf{inst}(\overrightarrow{\alpha:\kappa})}$$

(int) は整数型のコンストラクタの導出規則である。今、λ_{VL} のプリミティブ型は Int 型の身であるため、コンストラクタの導出規則はこれのみである。なお、n は任意の整数の値を表すメタ変数である。

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \quad D; \Sigma; \Gamma \vdash t : A}{D; \Sigma; \Gamma + [\Delta]_{0:R} \vdash t : A}\ (\textsc{weak}) \qquad \frac{\Sigma \vdash R : \mathsf{Coeff} \quad D; \Sigma; \Gamma, x : A \vdash t : B}{D; \Sigma; \Gamma, x : [A]_{1:R} \vdash t : B}\ (\textsc{der}) \qquad \frac{\Sigma \vdash r : \uparrow R \quad \Sigma \vdash R : \mathsf{Coeff} \quad D; \Sigma; [\Gamma] \vdash t : A}{D; \Sigma; r \cdot [\Gamma] \vdash [t] : \Box_r A}\ (\textsc{pr})$$

続く (weak)・(der)・(pr) は Brunel のコエフェクト計算 $\ell\mathcal{R}$PCF[19] 由来の構文である。各規則が弱化・自明な版リソースの導入・自由な版リソースの導入に対応しており、カインド導出により与えられた Version 型 R(現段階では λ_{VL} の Coeff は Version のみ) に用いることができる。注意するべきは (pr) 規則の結論の項変数環境である。プログラムに版リソースを導入するとき、その項変数環境にも全て同じ版リソースを要求する。この演算を定めるのが以下の環境への版リソースの乗算 $\cdot$ である。以下では、型環境のうち、バージョン付き型の仮定のみが含まれるものを特別に $[\Gamma]$ と表し、コエフェクト付き環境と呼ぶ。

定義 4.3.5 [項変数環境への版リソースの乗算 $\cdot$]
項変数環境 Γ について、Γ がコエフェクト付き環境であるとする。このとき、Γ へのコエフェクト $r \in R$ の乗算 $r \cdot \Gamma$ を以下のように定義する。

$$r \cdot \emptyset = \emptyset \qquad r \cdot (\Gamma, x : [A]_s) = (r \cdot \Gamma), x : [A]_{(r \otimes s)}$$

最後の 4 つの規則は λ_{VL} 由来の構文である。

$$\frac{D; \Sigma; [\Gamma_i] \vdash t_i : A}{D; \Sigma; \bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \{\overline{l = t} | l_i\} : \Box_{\overline{l}} A}\ (\textsc{ver}) \qquad \frac{D; \Sigma; [\Gamma_i] \vdash t_i : A}{D; \Sigma; \bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \langle \overline{l = t} | l_i \rangle : A}\ (\textsc{veri})$$

(ver) と (veri) はそれぞれ Versioend Value と中間項の型付け規則である。(ver) は各バージョン固有の定義に対応する項 t_i それぞれが型付け可能なとき、結論の Versioned Value の型に $\Box_{\overline{l}} A$ を与える。なお、このとき

λ_{VL} の縮約規則

$$\frac{}{(t \rhd \_)t = t}\ (\rhd\_) \qquad \frac{}{(t \rhd x)t = [t/x]t}\ (\rhd_{\mathrm{var}})$$

$$\frac{(t \rhd x)t = t'}{([t] \rhd [x])t = t'}\ (\rhd_{\square}) \qquad \frac{}{(\{\overline{l = t} \mid l_i\} \rhd [x])t = [\langle \overline{l = t} \mid l_i \rangle / x]t'}\ (\rhd_{\mathrm{ver}})$$

λ_{VL} の評価規則

$$\frac{t_1 \rightsquigarrow t_1'}{t_1\ t_2 \rightsquigarrow t_1'\ t_2}\ (\textsc{app}) \qquad \frac{t \rightsquigarrow t'}{t.l \rightsquigarrow t'.l}\ (\textsc{rstr}) \qquad \frac{}{(\lambda p.t)t' \rightsquigarrow (t' \rhd p)t}\ (\mathrm{p}\beta)$$

$$\frac{}{[t].l \rightsquigarrow t@l}\ (\textsc{extr1}) \qquad \frac{}{\{\overline{l = t} | m\}.l_i \rightsquigarrow t_i@l_i}\ (\textsc{extr2}) \qquad \frac{}{\langle \overline{l = t} | l_i \rangle \rightsquigarrow t_i@l_i}\ (\textsc{ver})$$

λ_{VL} のデフォルトバージョン上書き規則

$$n@l = n \qquad (\lambda x.t)@l = \lambda x.(t@l) \qquad [t]@l = [t]$$

$$(t\ u)@l = (t@l)\ (u@l) \qquad \{\overline{l = t} | l_i\}@l' = \{\overline{l = t} | l_i\} \qquad (t.l)@l' = (t@l').l$$

$$\frac{l' \in \{\bar{l}\}}{\langle \overline{l = t} | l_i \rangle @l' = \langle \overline{l = t} | l' \rangle} \qquad \frac{l' \notin \{\bar{l}\}}{\langle \overline{l = t} | l_i \rangle @l' = \langle \overline{l = t} | l_i \rangle}$$

図 4.4 λ_{VL} の縮約規則と評価規則

デフォルトバージョンはこの Versioned Value が使用可能なバージョンタグの中から選ばれる。(VERI) も (VER) と似たような型規則を持つが、項変数環境に要求するバージョンが異なる。これは、中間項は各バージョンの計算がデフォルトバージョンのみの計算を表すためである。

$$\frac{D; \Sigma; \Gamma \vdash t : \square_r A \quad l \in r}{D; \Sigma; \Gamma \vdash t.l : A}\ (\textsc{rstr}) \qquad \frac{D; \Sigma; \Gamma, x : [A]_r, \Gamma' \vdash t : B \quad r \sqsubseteq s}{D; \Sigma; \Gamma, x : [A]_s, \Gamma' \vdash t : B}\ (\sqsubseteq)$$

(RSTR) は (PR) や (VER) と対になっており、Versioned Value から特定のバージョン固有の計算を取り出す抽出演算子 $t.l$ の型付け規則である。指定されるバージョンタグ l は使用可能なバージョン中に存在していなければならない。($\sqsubseteq$) は版リソース代数に定義された順序に基づく部分型付け規則である。

4.4 λ_{VL} の意味論

λ_{VL} の意味論は名前呼びベースの遅延評価戦略を取る。λ_{VL} の値は以下のように与えられる。

$$(\text{値}) \quad v \quad ::= \quad x \mid n \mid [t] \mid \lambda p.t \mid \{\overline{l = t} \mid l\} \mid [t]$$

縮約中、値は $(v \rhd p)\ t = t'$ によって与えられたパターンと照合される。値 v はパターン p と照合され、t に値を代入して t' を生成する。また項の評価は、関係 $t \rightsquigarrow t'$ によって定義される。ここで (Pβ) で関数適用を評価する際は縮約規則により代入される項を生成する。また、(EXTR1)・(EXTR2) でバジョンタグ l のバージョン固有の定義を抽出する際は、デフォルトバージョン上書き規則によりサスペンド演算子 $[t]$ と Versioned Value 以外

のバージョンタグを揃えることで評価を進める。

λ_{VL} は Versioned Value と $[t]$ の評価をサスペンドしなければならないという制約があるため、必然的に名前呼び意味論を選択したが、いくつかのコエフェクト計算で評価戦略は異なるものを採用している。λ_{VL} のベースとなった $\ell\mathcal{R}$PCF[19] は λ_{VL} と同じく名前呼び意味論を採用している。$\ell\mathcal{R}$PCF でもプロモーション $[t]$ は値であり、縮約が一時停止されている。[22] も名前呼び意味論ベースの等式理論を示した。一方 Granule[23] はエフェクト計算を扱えるコエフェクト計算であり、副作用の発生順の混乱を避ける為値呼びの意味論を採用している。

4.5 型安全性に関する議論

最後に、λ_{VL} の型安全性について述べる。大枠の議論は Granule の型安全性に関する議論と同様であるが、λ_{VL} には版リソースの特別な導入規則が存在することに注意しなければならない。詳細な証明については付録 A.2, A.3, A.4 で論じる。

補題 4.5.1 [線形変数への代入補題]

$D;\Sigma;\Delta \vdash t' : A$ かつ $D;\Sigma;\Gamma, x : A, \Gamma' \vdash t : B$ であるならば、$D;\Sigma;\Gamma + \Delta + \Gamma' \vdash [t'/x]t : B$

証明

$D;\Sigma;\Delta \vdash t' : A$ の型導出に関する構造的帰納法 □

補題 4.5.2 [コエフェクト付き変数への代入補題]

$D;\Sigma;[\Delta] \vdash t' : A$ かつ $D;\Sigma;\Gamma, x : [A]_r, \Gamma' \vdash t : B$ ならば、$D;\Sigma;\Gamma + r \cdot \Delta + \Gamma' \vdash [t'/x]t : B$

証明

$D;\Sigma;[\Delta] \vdash t' : A$ の型導出に関する構造的帰納法 □

補題 4.5.3 [線形パターンに関する型安全性]

パターン p と項 t, t'' についてある t' が存在し、以下が成立する。

$$\left.\begin{array}{r} D;\Sigma;- \vdash p : A \rhd \Delta \\ D;\Sigma;\Gamma_2 \vdash t'' : A \\ D;\Sigma;\Gamma_1, \Delta \vdash t : B \end{array}\right\} \implies \exists t'. \begin{cases} (t'' \rhd p)t = t' & \text{（進行性）} \\ D;\Sigma;\Gamma_1 + \Gamma_2 \vdash t' : B & \text{（保存性）} \end{cases}$$

証明

$D;\Sigma;- \vdash p : A \rhd \Gamma$ の型導出に関する構造的帰納法 □

補題 4.5.4 [コエフェクト付きパターンに関する型安全性]

パターン p と項 t, t'' についてある t' が存在し、以下が成立する。

$$\left.\begin{array}{r} D;\Sigma;r : R \vdash p : A \rhd \Delta \\ D;\Sigma;[\Gamma_2] \vdash t'' : A \\ D;\Sigma;\Gamma_1, \Delta \vdash t : B \end{array}\right\} \implies \exists t'. \begin{cases} (t'' \rhd p)t = t' & \text{（進行性）} \\ D;\Sigma;\Gamma_1 + r \cdot \Gamma_2 \vdash t' : B & \text{（保存性）} \end{cases}$$

証明

$D;\Sigma;r : R \vdash p : A \rhd \Gamma$ の型導出に関する構造的帰納法 □

定理 4.5.5 [λ_{VL} の型安全性]

環境 D, Σ, Γ、項 t、型 A が存在して以下を満たす。

$$D; \Sigma; \Gamma \vdash t : A \Longrightarrow (\mathsf{value}\ t) \vee (\exists t', \Gamma'.t \rightsquigarrow t' \wedge D; \Sigma; \Gamma' \vdash t' : A' \wedge \Gamma' \sqsubseteq \Gamma)$$

証明

$D; \Sigma; \Gamma \vdash t : A$ の型導出に関する構造的帰納法 □

第 5 章

関連研究

5.1 依存関係の簡略化のための規範・技術

依存性地獄を回避するため、現代のソフトウェア開発では

- 各ライブラリの提供するバージョンを一つに定める
- 一つの環境に存在する各ライブラリのバージョン管理を厳格に行う

ことが一般的であり、依存関係を簡略化する為の規範・依存関係の管理をサポートする様々な技術がスタンダードとなっている。一方で前節で述べたように、依存性地獄の解決には処理系のレイヤーに互換性の有無をキャプチャする仕組みが必要なため、これらの規範・技術は根本的には解決法にはならない。

プラットフォーマーの対応

Android OS のような OS のアップデートには少ない数のセキュリティ関連のものが含まれるため、基本的には OS と共有ライブラリのアップデートは一体である。共有ライブラリのうち特定のアプリケーションの依存する部分はごく一部であり、本質的にはその局所的な依存部をアップデートから分離できればアップデート前の機能の一部を保存したままアップデートの恩恵を受けることは可能であるが、現代の多くのプラットフォームではこの選択肢を取らなかった。Android OS や iOS をはじめとする多くのプラットフォームでは、非互換性に依拠するユーザーのトラブルを回避するため、エコシステム全体で最新の共有ライブラリを使い続けるという選択をした。この目的を達成するため、プラットフォーマーは公式ストアで配信されるアプリケーションのターゲット API レベルに厳しい要件を課している。iOS では、2020 年 4 月以降 Apple に提出されるアプリケーションに、最新版の共有ライブラリである iOS 13 SDK でのビルドが可能であることを要件としている。[24] また Android OS では、2019 年 11 月以降の Google Play Console のアプリ提供者に、新規アプリのみならず既存アプリも最新の Android API に対応させる義務を負わせている [25]。

パッケージマネージャ

パッケージマネージャは環境中に存在するライブラリのバージョン管理の一元化を目的としたソフトウェアであり、2000 年代以降のプロダクトで採用されるプログラミング言語の殆どがパッケージマネージャを備えている。例えば Python には poetry[26]、Ruby には gem[27] と bundler[28]、C++ には conan[29] といったものが存在する。これらのパッケージマネージャはいずれもライブラリ単位でプログラムを管理し、環境へのインストールや削除を簡易に行うことができる。また、多くは新たなライブラリのインストール時に適切なバージョン

を自動で選択（依存性解決）することが出来る。また、最近の幾つかの最近の幾つかの洗練されたビルドシステム（Rust の cargo[30], Node.js の npm[4] 等）はパッケージマネージャと連携し、Dependency Hell の部分的な解決を目的とした機能を備えている。これらのビルドシステムは依存性解決を行うだけでなく、依存性グラフを部分グラフに分割し、部分グラフ毎に各ライブラリの適切なバージョンを選択できる機能を備えている。しかしいずれのシステムも用いても、衝突する依存性パスを合理的に共存させることは不可能である。パッケージマネージャは衝突する依存性パスの発見をサポートするが、見つかった場合は依然として一つの依存性パスを選択しなければならない。

コンテナ技術

コンテナ技術はパッケージマネージャとは異なり、環境単位で依存関係を管理することができる。中でも Docker[31] は、依存性地獄対策としてコンテナ技術を利用していることを明確にしている。Docker プロジェクトの目標は、全ての依存関係と共に環境毎アプリケーションをパッケージ化し、異なる開発・テスト・及び労働環境でスムーズに実行することである。近年のアプリケーションは、多くの場合既存アプリケーションの組み合わせで実現される。しかし、それぞれのアプリケーションには衝突する依存性パスを持つ可能性のある別個の依存関係が存在するため、一つの環境で関連する全てのアプリケーションの開発を行うことは困難であった。Docker はこのようなケースで威力を発揮する。Docker を利用することで、アプリケーション毎にコンテナとして環境を隔離することができるため、一つの物理マシンの中でバージョン違いのライブラリを共存させることが可能になる。また、全ての依存関係は Docker コンテナ内に同梱されているため、別の物理マシンに Docker コンテナを移植しても依存ライブラリの欠落が起きることはない。従って開発環境の移行が極めて容易である。しかし一方で、Docker プロジェクトの関心はコンテナ内部の依存関係には向いていない。すなわち、一つのアプリケーションの依存性グラフ内に衝突する依存性パスが存在する場合は、依然として依存性地獄が発生する。

モノレポジトリ方式

モノレポジトリ方式とは、バージョン管理システム上で通常は別のレポジトリに存在する二つ以上の論理ライブラリを、一つのレポジトリでバージョン管理する手法である。一つのレポジトリで管理することで、内包する全てのライブラリに単一のバージョン管理を行うことが可能になる。モノレポジトリ方式により複数ライブラリに渡るアップデートを単一のコミットで管理でき、ライブラリ間の依存関係を一元的に把握し、ライブラリ間にわたるテストやバグの発見を簡単にできるメリットがある。一方で、ライブラリを取り込む場合と同様にモジュール性が低下するため、一部の機能には本質的に関係ない膨大な数のコミットが発生したり、管理すべきファイル数の爆発的な増加によるバージョン管理システムの性能の低下が指摘されている。モノレポジトリ方式は開発の規模が大きいコミュニティで特に有用であるとされており、Google[32] や Facebook[33] が社内ライブラリをモノレポジトリで管理していることを公表している。メリットの一方で、モノレポジトリ方式も依存関係の把握を助ける為のテクニックに過ぎず、依存性地獄を解決する技術ではない。モノレポジトリの中に存在する多数のライブラリ間の依存関係が正当なものであるかは別の手法で検証しなければならないし、モノレポジトリの管理外にあるライブラリについては、依然として依存性地獄が発生する。

5.2 コエフェクト計算

コエフェクト計算は、線形型システム [8, 9] がベースとなった、コエフェクトと呼ばれる様々な計算的資源の使用状況を分析するためのフレームワークである。線形型システムはデータを線形リソースとして扱う。線形リソースとは、一度使う必要がある一方で、二度と使用することは出来ないデータのことである。例えば、恒等関数 $\lambda x.x$ は変数 x を束縛した後一回だけ使われるため、線形型システムで型付け可能である。しかし、$\lambda x.\lambda y.x$ は束縛された y が使用されず、$\lambda x.x\ x$ は束縛した変数 x が二回使われているため、これらの関数は線形型システムで型付けできない。線形型システムにおいて、線形リソースの対になる概念が非線形リソースである。非線形リソースはモーダル演算子！によってキャプチャされ、使用回数に関する制約が解除されたことを明示できる。ただし、多くのプログラミング言語では殆どの値が非線形リソースであり、その上線形型システムでキャプチャできる計算的リソースの使用状況はあまり詳細なものではなかった。有界線形論理は線形論理の！モダリティを一般化することでこの難点を克服した。有界線形論理では！でキャプチャされたリソースの使用量の上限を定量化することができ、例えば $!_n A$ 型は、最大で n 回使用できる A 型の値をキャプチャする。

2010 年以降、有界線形論理の！モダリティのキャプチャ対象となる計算的リソースの一般化が試みられてきた。Dal Lago と Gaboardi らは、使用回数についてインデックス付けされたモダリティをキャプチャできる計算体系 Linear PCF[34, 35] を提案した。また、De Amorim らは値に与えられるプライバシー情報をキャプチャ可能な計算体系 DFuzz[36] を提案した。また著者らはプログラムのバージョン依存性をキャプチャ可能な計算体系 λ_{VL} [1] を提案した。

モダリティのキャプチャ対象となる計算的リソースの一般化とほぼ同時期に、2013 年から 2014 年頃コエフェクトの概念が考案された。Petricek らはエフェクト計算の圏論的双対としてコエフェクト計算 [37] を定義し、Brunel ら有界線形論理の一般化としてコエフェクト計算 $\ell\mathcal{R}$PCF[19] を定義した。このようにどちら別の背景を持ち独立に生まれた研究であったが、各型システムは基本的に同じ構造を持つ。キャプチャ対象の計算的リソースを代数的構造によって抽象化し、変数に与えられた計算的リソースをデータフローと伴に追跡することにより、そのプログラムがどのような文脈で評価されるべきかを解析する。λ_{VL} の元となった $\ell\mathcal{R}$PCF は有界線形論理における自然数のインデックスを半環に置き換えた。そのためプログラムの型に明示的に計算的リソースのモダリティが与えられている。一方で Petriceik らの型システムにおいては型のモダリティは暗黙的であり、項変数文脈においては保持されているが、型には直接的に現れない。

直近の研究ではコエフェクト計算とエフェクト計算を統一的に表現するフレームワークが研究されてきた。Gaboardi と Katsumata らはエフェクト計算とコエフェクト計算を Graded Modal Type によって統一することを検討した [22]。Orchard らはそれに標準的なプログラミング言語機能 (カインドベースの多相性・パターンマッチ・型強制・代数的データ構造・型制約) を導入した計算体系 Gr[20] を提案した。Gr は $\ell\mathcal{R}$PCF と類似のコエフェクト計算 `GrMini`(単純型付きラムダ計算レベルのもの) を計算の核として持ち、言語処理系として Granule を持つ。本研究では `GrMini` から `Gr` への言語拡張を参考にし、λ_{VL} への型・版多相性の導入を行った。

第 6 章

結論と今後の研究

本研究では、計算的資源としてバージョンを扱ったコエフェクト計算である λ_{VL} を提案した。これにより従来プログラミング言語の外（パッケージマネージャやビルドツール）で管理されていたライブラリのバージョンに関する関心事をプログラミング言語の内部に持ち込み、型検査により検証可能にした。さらに、λ_{VL} にカインドベースの型・版多相性を導入した。型多相性は λ_{VL} にモジュール機構を導入する契機になり、版多相性はプログラムからのバージョン依存性の完全な排除と、バージョンに関する型レベル制約の記述の第一歩となる。将来の課題のうち大きなものは以下の 2 つである。

6.1 本格的なモジュール言語

本研究ではバージョンの概念を取り込んだモジュール言語を設計することを目的としている為、どのような形式でモジュール言語を設計するかは検討課題の一つである。1980 年代から盛んに研究が行われてきた ML 式のモジュール機構 [21] は、コア言語（型と式）とモジュール言語（signature, structure, functor）からなるものであり、モジュール言語はコア言語に上に独立の言語として設計されている。コア言語の中で扱えるモジュールは第一級モジュールと呼ばれ、型検査を決定不能にしてしまうこと [38] が知られている。

一方で近年の研究ではモジュールをコアレベルで操作できるよう、モジュール言語とコア計算を統合するよう再設計した ML 言語 1ML[13, 39] も提案されている。1ML は関数・ファンクター・型コンストラクタの全てが同じレイヤーで扱える言語であり、本質的に F_ω(カインド拡張された System F) と等価である。1ML は既にコエフェクト計算の圏論的双対であるエフェクト計算との統合が検討されており [40]、コエフェクト計算との統合に良い見通しがある。

6.2 バージョン制約式の導入

本研究ではトップレベル定義は型スキームであった。一方で本研究の型・版多相性の拡張のベースとなった `Gr`[20] では、トップレベル定義にコエフェクトの制約条件を含めることが可能であり、制約条件の為の構文・型システムが用意されている。この拡張を λ_{VL} に導入することで、λ_{VL} でも型レベルにバージョンに関する制約を含めることが可能となる。

例えば「$v18$ 以下の AlarmManager には完全な互換性があり、一方で $v19$ 以降の AlarmManager とは非互換である」というライブラリの性質が何らかの方法で提供されているとき、「この関数の引数は $v18$ 以下の AlarmManager 型の値であれば何でもよい」というような型レベルの制約を記述できると便利である。コエフェ

クトシステムは計算的リソースにある種の計算が可能かつ順序構造を持つことを要求する為、λ_{VL} は既に前者のようなバージョンに関する互換性の性質を記述する為の手段は備えていた。一方で後者のような制約を型レベルで記述するためには、

- バージョンを r とパラメータ化できる機構
- $r \leq \upsilon 18$ に相当する制約を記述可能な型の構文
- 制約をキャプチャできる型システム

の 3 つを λ_{VL} に導入する必要がある。第 3.2.2 節で論じたように、本研究の版多相性の導入により、このうち 1 つ目が実現された。残りの 2 つは将来の課題である。

参考文献

[1] Yudai Tanabe, Tomoyuki Aotani, and Hidehiko Masuhara. A context-oriented programming approach to dependency hell. In *Proceedings of the 10th International Workshop on Context-Oriented Programming: Advanced Modularity for Run-time Composition*, COP '18, pp. 8–14, New York, NY, USA, 2018. ACM.

[2] Android Developpers - Platform. https://developer.android.com/about/.

[3] browserify. http://browserify.org/.

[4] npm | building amazing things. https://www.npmjs.com/.

[5] Visualization of npm dependencies. http://npm.anvaka.com/#/view/2d/browserify.

[6] David B. MacQueen. Using dependent types to express modular structure. In *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '86, p. 277–286, New York, NY, USA, 1986. Association for Computing Machinery.

[7] Barbara H. Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, Vol. 16, No. 6, p. 1811–1841, November 1994.

[8] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, Vol. 50, No. 1, p. 1–102, January 1987.

[9] Philip Wadler. Linear types can change the world! In *PROGRAMMING CONCEPTS AND METHODS*. North, 1990.

[10] J. Y. Girard. Une extension de l'interpretation de Godel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. Vol. 63, pp. 63–92, 1971.

[11] John C. Reynolds. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque Sur La Programmation*, p. 408–423, Berlin, Heidelberg, 1974. Springer-Verlag.

[12] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Trans. Program. Lang. Syst.*, Vol. 10, No. 3, pp. 470–502, July 1988.

[13] Andreas Rossberg, Claudio V. Russo, and Derek Dreyer. F-ing modules. In *Proceedings of the 5th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, TLDI '10, p. 89–102, New York, NY, USA, 2010. Association for Computing Machinery.

[14] David MacQueen. Modules for standard ml. In *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84, pp. 198–207, New York, NY, USA, 1984. ACM.

[15] Robert Harper and Mark Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *Proceedings of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '94, pp. 123–137, New York, NY, USA, 1994. ACM.

[16] Xavier Leroy. A syntactic theory of type generativity and sharing. *J. Funct. Program.*, Vol. 6, No. 5, pp. 667–698, 1996.

[17] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 1st edition, 2002.

[18] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, Vol. 5, No. 3, pp. 223–255, December 1977.

[19] Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A Core Quantitative Coeffect Calculus. In *Programming Languages and Systems*, Lecture Notes in Computer Science, pp. 351–370. Springer, Berlin, Heidelberg, April 2014.

[20] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.*, Vol. 3, No. ICFP, July 2019.

[21] Robin Milner, Mads Tofte, and David Macqueen. *The Definition of Standard ML*. MIT Press, Cambridge, MA, USA, 1997.

[22] Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. Combining Effects and Coeffects via Grading. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, pp. 476–489, New York, NY, USA, 2016. ACM.

[23] Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proc. ACM Program. Lang.*, Vol. 3, No. ICFP, July 2019.

[24] Apple Developpers - Submit Your Apps to the App Store. https://developer.apple.com/ios/submit/.

[25] Android Developpers - Expanding target API level requirements in 2019. https://android-developers.googleblog.com/2019/02/expanding-target-api-level-requirements.html, February 2019.

[26] Poetry - Python dependency management and packaging made easy. https://python-poetry.org/.

[27] RubyGems.org. https://rubygems.org/.

[28] Bundler: The best way to manage a Ruby application's gems. https://bundler.io/.

[29] Conan, the C / C++ Package Manager for Developers. https://conan.io/.

[30] Introduction: The Cargo Book. https://doc.rust-lang.org/cargo/.

[31] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, Vol. 2014, No. 239, March 2014.

[32] Rachel Potvin and Josh Levenberg. Why google stores billions of lines of code in a single repository. *Commun. ACM*, Vol. 59, No. 7, p. 78–87, June 2016.

[33] Durham Goode. Facebook Engineering: Scaling Mercurial at Facebook. https://code.fb.com/core-data/scaling-mercurial-at-facebook/, January 2014.

[34] Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *Proceedings of the 2011 IEEE 26th Annual Symposium on Logic in Computer Science*, LICS '11, p. 133–142, USA, 2011. IEEE Computer Society.

[35] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, p. 357–370, New York, NY, USA, 2013. Association for Computing Machinery.

[36] Arthur Azevedo de Amorim, Marco Gaboardi, Emilio Jesús Gallego Arias, and Justin Hsu. Really natural linear indexed type checking. In *Proceedings of the 26nd 2014 International Symposium on Implementation and Application of Functional Languages*, IFL '14, New York, NY, USA, 2014. Association for Computing

Machinery.

[37] Tomas Petricek, Dominic Orchard, and Alan Mycroft. Coeffects: A Calculus of Context-dependent Computation. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pp. 123–135, New York, NY, USA, 2014. ACM.

[38] Mark Lillibridge. Translucent sums: A foundation for higher-order module systems. 08 1997.

[39] Andreas Rossberg. 1ml – core and modules united (f-ing first-class modules). *SIGPLAN Not.*, Vol. 50, No. 9, p. 35–47, August 2015.

[40] Andreas Rossberg. *1ML with Special Effects*, Vol. 9600, pp. 336–355. 03 2016.

付録 A

証明

A.1 型代入の定義

定義 A.1.1 **[型代入 θ]**

型代入 $\theta = (\alpha \mapsto A)$ が与えられたとき、型・コエフェクト・カインド・項変数環境・型変数環境・型代入への各型代入を以下のように定義する。

型への型代入

$$
\begin{aligned}
\theta K &= K \\
\theta \alpha &= A \quad (\theta(\alpha) = A) \\
\theta \alpha &= \alpha \quad (\text{otherwise}) \\
\theta(A \to B) &= \theta A \to \theta B \\
\theta(A\ B) &= (\theta A)\ (\theta B) \\
\theta(A \operatorname{op} B) &= (\theta A) \operatorname{op} (\theta B) \\
\theta(\Box_r A) &= \Box_{(\theta r)}(\theta A)
\end{aligned}
$$

コエフェクトへの型代入

$$
\begin{aligned}
\theta 0 &= 0 \\
\theta 1 &= 1 \\
\theta \alpha &= A \quad (\theta(\alpha) = A) \\
\theta \alpha &= \alpha \quad (\text{otherwise}) \\
\theta(r_1 \otimes r_2) &= (\theta r_1) \otimes (\theta r_2) \\
\theta(r_1 \oplus r_2) &= (\theta r_1) \oplus (\theta r_2)
\end{aligned}
$$

カインドへの型代入

$$
\begin{aligned}
\theta\ \uparrow A &= \uparrow(\theta A) \\
\theta(\kappa_1\ \kappa_2) &= (\theta \kappa_1)\ (\theta \kappa_2) \\
\theta \kappa &= \kappa
\end{aligned}
$$

項変数環境への型代入

$$
\begin{aligned}
\theta \emptyset &= \emptyset \\
\theta(\Gamma, x : A) &= \theta\Gamma, x : \theta A \\
\theta(\Gamma, x : [A]_r) &= \theta\Gamma, x : [(\theta A)]_{(\theta r)}
\end{aligned}
$$

型変数環境への型代入

$$
\begin{aligned}
\theta \emptyset &= \emptyset \\
\theta(\Sigma, \alpha : \kappa) &= \theta\Sigma, \alpha : \theta\kappa
\end{aligned}
$$

型代入への型代入

$$
\begin{aligned}
\theta \emptyset &= \emptyset \\
\theta(\theta' \uplus \alpha \mapsto A) &= (\theta\theta') \uplus \alpha \mapsto (\theta A)
\end{aligned}
$$

A.2 項変数環境に関する補題

はじめにコエフェクト付き項変数環境の操作に関する定義・補題について述べる。

定義 A.2.1 [環境の結合 +]

交わりの無い 2 つの環境は結合可能である。さらに両方の環境に現れるコエフェクト付きの仮定は、構造的半環の加算 $\oplus$ を用いて結合可能である。このとき、2 つの環境の結合 + を以下のように定義する。

$$(\Gamma, x:A) + \Gamma' = (\Gamma + \Gamma'), x:A \quad \text{iff } x \notin |\Gamma'| \qquad \emptyset + \Gamma = \Gamma$$
$$\Gamma + (\Gamma', x:A) = (\Gamma + \Gamma'), x:A \quad \text{iff } x \notin |\Gamma| \qquad \emptyset + \Gamma = \Gamma$$
$$(\Gamma.x:[A]_r) + (\Gamma', x:[A]_s) = (\Gamma + \Gamma'), x:[A]_{(r \oplus s)}$$

また、以上で定義した + を用い、項変数環境の総和を以下のように定める。

$$\bigcup_{i \in n} \Gamma_i = \Gamma_1 + \cdots + \Gamma_n$$

ここで注意しなければならないのは、2 つの項変数環境の結合演算 , と + の違いである。「Γ_1, Γ_2」と表記された場合、$\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$（$\Gamma_1$ と Γ_2 は非交）、すなわち Γ_1 と Γ_2 は同じ項変数についての仮定を含まない。一方で「$\Gamma_1 + \Gamma_2$」と表記された場合、線形な項変数についての仮定 $x:A$ (線形仮定) は Γ_1 と Γ_2 のどちらか一方のみにしか含まれないが、非線形な項変数についての仮定 $x:[A]_r$ (コエフェクト仮定) については Γ_1 と Γ_2 の両方に含まれうる。この性質は代入補題の証明において重要となる。なお、コエフェクト仮定のみが含まれる項変数環境を特別にコエフェクト付き環境と呼び、$[\Gamma]$ で表す。

定義 A.2.2 [項変数環境への版リソースの乗算 ·]

項変数環境 Γ について、Γ がコエフェクト付き環境であるとする。このとき、Γ へのコエフェクト $r \in R$ の乗算 $r \cdot \Gamma$ を以下のように定義する。

$$r \cdot \emptyset = \emptyset \qquad r \cdot (\Gamma, x:[A]_s) = (r \cdot \Gamma), x:[A]_{(r \otimes s)}$$

定義 A.2.3 [項変数環境の特徴付け]

項変数環境 Γ_1, Γ_2 について、Γ_2 で特徴付けられた Γ_2 の部分列 $\Gamma_{1|\Gamma_2}$ と $\Gamma_{1|\overline{\Gamma_2}}$ を以下のように定義する。

$$\Gamma_{1|\Gamma_2} := \{x:A \mid x \in \mathrm{dom}(\Gamma_1) \wedge x \in \mathrm{dom}(\Gamma_2)\}$$
$$\Gamma_{1|\overline{\Gamma_2}} := \{x:A \mid x \in \mathrm{dom}(\Gamma_1) \wedge x \notin \mathrm{dom}(\Gamma_2)\}$$

すなわち、$\Gamma_{1|\Gamma_2}$ は Γ_2 に含まれる項変数を全て含む Γ_1 の部分列であり、$\Gamma_{1|\overline{\Gamma_2}}$ は Γ_1 のうち Γ_2 に含まれない項変数を全て含む Γ_1 の部分列である。

$\Gamma_{1|\Gamma_2}$ と $\Gamma_{1|\overline{\Gamma_2}}$ を用いて、項変数環境の等式変形に関数するいくつかの補題を述べる。なお、これらの補題は A.2.3 の定義から直ちに成立する。

補題 A.2.4 [項変数環境の特徴づけによる分割]

項変数環境 Γ_1, Γ_2 について以下が成立する。

$$\Gamma_{1|\Gamma_2} + \Gamma_{1|\overline{\Gamma_2}} = \Gamma_1$$

補題 A.2.5 [項変数環境の並び替え]

項変数環境 $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ と仮定 $x : A$ について、以下の並び替えが成立する。

$$(\Gamma_1, x : A, \Gamma_1') + \Gamma_2 = (\Gamma_1 + \Gamma_{2|\Gamma_1}), x : A, (\Gamma_1' + \Gamma_{2|\overline{\Gamma_1}}) \tag{1}$$

$$\Gamma_1 + (\Gamma_2, x : A, \Gamma_2') = (\Gamma_{1|\overline{\Gamma_2'}} + \Gamma_2), x : A, (\Gamma_{1|\Gamma_2'} + \Gamma_2') \tag{2}$$

$$((\Gamma_1, \Gamma_2) + (\Gamma_3, \Gamma_4)) = ((\Gamma_1 + \Gamma_{3|\Gamma_1} + \Gamma_{4|\Gamma_1}), (\Gamma_2 + \Gamma_{3|\overline{\Gamma_1}} + \Gamma_{4|\overline{\Gamma_1}})) \tag{3}$$

補題 A.2.6 [項変数環境の結合とコエフェクト乗算の分配則]

項変数環境 Γ とコエフェクト $r_1, r_2 \in R$ について、以下の等式が成立する。

$$(r_1 \cdot \Gamma) + (r_2 \cdot \Gamma) = (r_1 \otimes r_2) \cdot \Gamma$$

補題 A.2.7 [項変数環境 Δ の非交分割]

項変数環境 $\Gamma_1, \Delta, \Gamma_2$ について、以下の等式が成立する。

$$(\Gamma_1, \Delta, \Gamma_2) = (\Gamma_1 + \Delta_{|\Gamma_1}), \Delta_{|\overline{(\Gamma_1, \Gamma_2)}}, (\Gamma_2 + \Delta_{|\Gamma_2})$$

A.3 代入補題

線形変数への代入補題と、コエフェクト付き変数への代入補題のそれぞれについて証明する。

補題 A.3.1 [線形変数に関する代入補題]

$D; \Sigma; \Delta \vdash t' : A$（仮定 1）かつ $D; \Sigma; \Gamma, x : A, \Gamma' \vdash t : B$（仮定 2）であるならば、$D; \Sigma; \Gamma + \Delta + \Gamma' \vdash [t'/x]t : B$

証明

$D; \Sigma; \Gamma, x : A, \Gamma' \vdash t : B$（仮定 2）の型導出に関する構造的帰納法で示す。仮定 2 の型導出の最後に用いた規則で場合分け。

$$\text{Case. } \frac{\Sigma \vdash B : \mathsf{Type}}{D; \Sigma; y : B \vdash y : B}\ (\textsc{var})$$

最後の導出規則が (VAR) であったとき、以下が成立する。

- $\Gamma = \Gamma' = \emptyset$
- $x = t = y$
- $B = A$

従って示すべき補題の結論は以下のようになる。

$$D; \Sigma; \Delta \vdash [t'/y]y : B$$

代入の定義から $[t'/y]y = t'$ である為、補題の結論は補題の仮定 1 そのものとなる。

$$\text{Case. } \frac{\begin{array}{c} D; \Sigma; - \vdash p : B_1 \rhd \Delta' \\ D; \Sigma; \Gamma, x : A, \Gamma', \Delta' \vdash t : B_2 \end{array}}{D; \Sigma; \Gamma, x : A, \Gamma' \vdash \lambda p.t : B_1 \to B_2}\ (\textsc{abs})$$

最後の導出規則が (ABS) のとき、帰納法の仮定から (ABS) の部分導出に補題を適用し、以下を得る。

$$D; \Sigma; \Gamma + \Delta + (\Gamma', \Delta') \vdash [t'/x]t : B_2$$

ここで Δ' は p 中で得られた変数束縛のみを含み、Γ, Δ, Γ' と非交でなければならない。従って直上の型導出式は以下と等しい。

$$D; \Sigma; (\Gamma + \Delta + \Gamma'), \Delta' \vdash [t'/x]t : B_2$$

ここで改めて (ABS) を適用することで以下を得る。

$$\frac{D; \Sigma; - \vdash p : B_1 \rhd \Delta' \qquad D; \Sigma; (\Gamma + \Delta + \Gamma'), \Delta' \vdash [t'/x]t : B_2}{D; \Sigma; \Gamma + \Delta + \Gamma' \vdash \lambda p.[t'/x]t : B_1 \to B_2}\ (\textsc{abs})$$

代入の定義から $\lambda p.[t'/x]t = [t'/x](\lambda p.t)$ であり、これにより補題の結論を得る。

Case. $$\frac{D; \Sigma; \Gamma_1 \vdash t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2 \vdash t_2 : B_1}{D; \Sigma; \Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : B_2}\ (\textsc{app})$$

最後の導出規則が (APP) であったとき、以下が成立する。

- $t = t_1\ t_2$
- $B = B_2$
- $(\Gamma, x : A, \Gamma') = (\Gamma_1 + \Gamma_2)$

ここで 3 つ目の等式に注目すると、項変数環境の結合 $+$ の定義から、線形仮定 $x : A$ は項変数環境 Γ_1, Γ_2 のいずれか一方にのみ含まれる。

- $(x : A) \in \Gamma_1$ の場合
  改めて $\Gamma_1 = (\Gamma_1', x : A, \Gamma_1'')$ とおく。これを用い、最後の型導出は以下のように書き換えられる。

$$\frac{D; \Sigma; \Gamma_1', x : A, \Gamma_1'' \vdash t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2 \vdash t_2 : B_1}{D; \Sigma; (\Gamma_1', x : A, \Gamma_1'') + \Gamma_2 \vdash t_1\ t_2 : B_2}\ (\textsc{app})$$

  ここで補題と直上の導出で項変数環境に注目すると

$$\begin{aligned}(\Gamma, x : A, \Gamma') &= (\Gamma_1 + \Gamma_2) \\ &= (\Gamma_1', x : A, \Gamma_1'') + \Gamma_2 \\ &= (\Gamma_1' + \Gamma_{2|\Gamma_1'}), x : A, (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}}) && (\because \text{A.2.5 (1)})\end{aligned}$$

  最左辺と最右辺に注目すれば $\Gamma = (\Gamma_1' + \Gamma_{2|\Gamma_1'})$ と $\Gamma' = (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}})$ を得られる。
  次に帰納法の仮定を 2 つの前提それぞれに適用し、改めて (APP) を適用することで以下を得る。

$$\frac{D; \Sigma; \Gamma_1' + \Delta + \Gamma_1'' \vdash [t/x]t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2 \vdash [t/x]t_2 : B_1}{D; \Sigma; (\Gamma_1' + \Delta + \Gamma_1'') + \Gamma_2 \vdash ([t/x]t_1)\ ([t/x]t_2) : B_2}\ (\textsc{app})$$

ここで $([t/x]t_1)\ ([t/x]t_2) = [t/x](t_1\ t_2)$ に注意すれば、直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後にこの項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma + \Delta + \Gamma') = ((\Gamma_1 + \Delta + \Gamma_1'') + \Gamma_2)$ を示す。$\Gamma = (\Gamma_1' + \Gamma_{2|\Gamma_1'})$ と $\Gamma' = (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}})$ から

$$\begin{aligned}(\Gamma + \Delta + \Gamma') &= (\Gamma_1' + \Gamma_{2|\Gamma_1'}) + \Delta + (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}}) \\ &= (\Gamma_1' + \Delta + \Gamma_1'' + (\Gamma_{2|\Gamma_1'} + \Gamma_{2|\overline{\Gamma_1'}})) && (\because + \text{の可換性と結合性}) \\ &= (\Gamma_1' + \Delta + \Gamma_1'' + \Gamma_2) && (\because \text{A.2.4}) \\ &= ((\Gamma_1' + \Delta + \Gamma_1'') + \Gamma_2) && (\because + \text{の結合性})\end{aligned}$$

以上より補題の結論を得る。

- $(x : A) \in \Gamma_2$ の場合

  $\Gamma_2 = (\Gamma_2', x : A, \Gamma_2'')$ を用い、最後の型導出は以下のように書き換えられる。

$$\frac{D; \Sigma; \Gamma_1 \vdash t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2', x : A, \Gamma_2'' \vdash t_2 : B_1}{D; \Sigma; \Gamma_1 + (\Gamma_2', x : A, \Gamma_2'') \vdash t_1\ t_2 : B_2}\ (\textsc{app})$$

  以下の証明は $(x : A) \in \Gamma_1$ のケースと全く同様である。ただし、項変数環境の等式変形に A.2.5 (1) ではなく A.2.5 (2) を用いる。

$$\text{Case.}\ \frac{}{D; \Delta; \emptyset \vdash n : \mathsf{Int}}\ (\textsc{int}) \quad \& \quad \text{Case.}\ \frac{(x : \forall\{\overrightarrow{\alpha : \kappa}\}.A) \in D \quad \theta, \Sigma' = \mathsf{inst}(\overrightarrow{\alpha : \kappa})}{D; \Sigma, \Sigma'; \emptyset \vdash x : \theta A}\ (\text{C})$$

最後の導出規則が (INT), (C) であったとき、結論の項変数環境は空（$= \emptyset$）である。従って補題の結論は自明に成立する。

$$\text{Case.}\ \frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma_1, x : A, \Gamma_2 \vdash t : B}{D; \Sigma; (\Gamma_1, x : A, \Gamma_2) + [\Delta']_{0:R} \vdash t : B}\ (\textsc{weak})$$

最後の導出規則が (WEAK) であったとき、線形仮定 $x : A$ はコエフェクト付き項変数環境である $[\Delta']_{0:R}$ には含まれない。ここで補題と (WEAK) で項変数環境に注目すると

$$\begin{aligned}(\Gamma, x : A, \Gamma') &= (\Gamma_1, x : A, \Gamma_2) + [\Delta']_{0:R} \\ &= (\Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1}), x : A, (\Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}}) && (\because \text{A.2.5 (1)})\end{aligned}$$

最左辺と最右辺に注目すれば、$\Gamma = \Gamma_1 + [\Delta']_{0:R|\Gamma_1}$ と $\Gamma' = \Gamma_2 + [\Delta']_{0:R|\overline{\Gamma_1}}$ を得る。また帰納法の仮定から補題を前提に適用し、改めて (WEAK) を適用することで以下を得る。

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma_1 + \Delta + \Gamma_2 \vdash [t'/x]t : B}{D; \Sigma; (\Gamma_1 + \Delta + \Gamma_2) + [\Delta']_{0:R} \vdash [t'/x]t : B}\ (\textsc{weak})$$

直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後に、この項変数環境が補題の結論のものに等

しいこと、すなわち $(\Gamma_1 + \Delta + \Gamma_2) + [\Delta']_{0:R} = \Gamma + \Delta + \Gamma'$ を示す。$\Gamma = \Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1}$ と $\Gamma' = \Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}}$ から

$$\begin{aligned}(\Gamma + \Delta + \Gamma') &= (\Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1}) + \Delta + (\Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}}) \\ &= \Gamma_1 + \Delta + \Gamma_2 + ([\Delta']_{0:R\,|\Gamma_1} + [\Delta']_{0:R\,|\overline{\Gamma_1}}) && (\because + \text{の結合性と可換性}) \\ &= \Gamma_1 + \Delta + \Gamma_2 + [\Delta']_{0:R} && (\because \text{A.2.4}) \\ &= (\Gamma_1 + \Delta + \Gamma_2) + [\Delta']_{0:R} && (\because + \text{の結合性})\end{aligned}$$

以上より補題の結論を得る。

$$\text{Case.} \quad \frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma, x : A, \Gamma'', y : B_1 \vdash t : B_2}{D; \Sigma; \Gamma, x : A, \Gamma'', y : [B_1]_{1:R} \vdash t : B_2}\ (\textsc{der})$$

最後の導出規則が (DER) であったとき、線形仮定 $x : A$ はコエフェクト仮定である $y : [B_1]_{1:R}$ とは異なる。帰納法の仮定から、補題を前提に適用して以下を得る。

$$D; \Sigma; \Gamma + \Delta + (\Gamma'', y : B_1) \vdash [t'/x]t : B_2$$

ここで $y : B_1$ は Γ, Δ, Γ'' のいずれとも非交な線形仮定であり、$\Gamma + \Delta + (\Gamma'', y : B_1) = (\Gamma + \Delta + \Gamma''), y : B_1$ を満たすことから

$$D; \Sigma; (\Gamma + \Delta + \Gamma''), y : B_1 \vdash [t'/x]t : B_2$$

改めて (DER) を適用して以下を得る。

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; (\Gamma + \Delta + \Gamma''), y : B_1 \vdash [t'/x]t : B_2}{D; \Sigma; (\Gamma + \Delta + \Gamma''), y : [B_1]_{1:R} \vdash [t'/x]t : B_2}\ (\textsc{der})$$

最後に、$y : [B_1]_{1:R}$ は $\Gamma + \Delta + \Gamma''$ と非交であるから、$((\Gamma + \Delta + \Gamma''), y : [B_1]_{1:R}) = \Gamma + \Delta + (\Gamma'', y : [B_1]_{1:R})$ より

$$D; \Sigma; \Gamma + \Delta + (\Gamma'', y : [B_1]_{1:R}) \vdash [t'/x]t : B_2$$

以上より補題の結論を得る。

$$\text{Case.} \quad \frac{\Sigma \vdash r : \uparrow R \quad \Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; [\Gamma] \vdash t : B}{D; \Sigma; r \cdot [\Gamma] \vdash [t] : \square_r B}\ (\textsc{pr})$$

最後の導出規則が (PR) であったとき、結論の項変数環境はコエフェクト付き型環境 $r \cdot [\Gamma]$ であり、線形変数を含まない。従って補題の結論は自明に成立する。

$$\text{Case.} \quad \frac{D; \Sigma; [\Gamma_i] \vdash t_i : A}{D; \Sigma; \bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \{\overline{l = t} | l_i\} : \square_{\overline{l}} A}\ (\textsc{ver}) \quad \& \quad \text{Case.} \quad \frac{D; \Sigma; [\Gamma_i] \vdash t_i : A}{D; \Sigma; \bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \langle \overline{l = t} | l_i \rangle : A}\ (\textsc{veri})$$

最後の導出規則が (VER), (VERI) であったとき、結論の項変数環境はコエフェクト付き環境であるため、線形変数を含まない。従って補題の結論は自明に成立する。

$$\text{Case. } \frac{D; \Sigma; \Gamma \vdash t : \square_r A \quad l \in r}{D; \Sigma; \Gamma \vdash t.l : A} \text{ (RSTR)}$$

最後の導出規則が (RSTR) であったとき、帰納法の仮定から前提に補題を適用し、得られた導出式に改めて (RSTR) を適用することで直接補題の結論を得る。

$$\text{Case. } \frac{D; \Sigma; \Gamma, y : [B']_r, \Gamma' \vdash t : B \quad r \sqsubseteq s}{D; \Sigma; \Gamma, y : [B']_s, \Gamma' \vdash t : B} \text{ (}\sqsubseteq\text{)}$$

最後の導出規則が (⊑) であったとき、線形仮定 $x : A$ はコエフェクト仮定である $y : [B_1]_{1:R}$ とは異なり、$(x : A) \in \Gamma$ か $(x : A) \in \Gamma'$ のいずれか一方のみを満たす。いずれのケースであっても帰納法の仮定から前提に補題を適用し、得られた導出式に改めて (⊑) を適当することで補題の結論を得る。 □

補題 A.3.2 [コエフェクト付き変数に関する代入補題]

$D; \Sigma; [\Delta] \vdash t' : A$（仮定 1）かつ $D; \Sigma; \Gamma, x : [A]_r, \Gamma' \vdash t : B$（仮定 2）ならば、$D; \Sigma; \Gamma + r \cdot \Delta + \Gamma' \vdash [t'/x]t : B$

証明

$D; \Sigma; \Gamma, x : [A]_r, \Gamma' \vdash t : B$（仮定 2）の型導出に関する構造的帰納法で示す。仮定 2 の型導出の最後に用いた規則で場合分け。

$$\text{Case. } \frac{\Sigma \vdash B : \mathsf{Type}}{D; \Sigma; y : B \vdash y : B} \text{ (VAR)}$$

最後の導出規則が (VAR) であったとき、$x : [A]_r$ はコエフェクト仮定で $y : B$ は線形仮定のため $x \neq y$ であり、なおかつ $y : B$ 以外の項変数環境は空である。従って置換するべきコエフェクト付き変数は存在せず、補題の結論が成立する。

$$\text{Case. } \frac{\begin{array}{c} D; \Sigma; - \vdash p : B_1 \rhd \Delta' \\ D; \Sigma; \Gamma, x : [A]_r, \Gamma', \Delta' \vdash t : B_2 \end{array}}{D; \Sigma; \Gamma, x : [A]_r, \Gamma' \vdash \lambda p.t : B_1 \to B_2} \text{ (ABS)}$$

最後の導出規則が (ABS) のとき、帰納法の仮定から (ABS) の部分導出に補題を適用し、以下を得る。

$$D; \Sigma; \Gamma + r \cdot \Delta + (\Gamma', \Delta') \vdash [t'/x]t : B_2$$

ここで Δ' は p 中で得られた変数束縛のみを含み、Γ, Δ, Γ' と非交でなければならない。従って直上の型導出式は以下と等しい。

$$D; \Sigma; (\Gamma + r \cdot \Delta + \Gamma'), \Delta' \vdash [t'/x]t : B_2$$

ここで改めて (ABS) を適用することで以下を得る。

$$\frac{D;\Sigma;-\vdash p:B_1\rhd\Delta' \qquad D;\Sigma;(\Gamma+r\cdot\Delta+\Gamma'),\Delta'\vdash[t'/x]t:B_2}{D;\Sigma;\Gamma+r\cdot\Delta+\Gamma'\vdash\lambda p.[t'/x]t:B_1\to B_2}\ (\textsc{abs})$$

代入の定義から $\lambda p.[t'/x]t=[t'/x](\lambda p.t)$ であり、これにより補題の結論を得る。

$$\text{Case.}\quad \frac{D;\Sigma;\Gamma_1\vdash t_1:B_1\to B_2 \qquad D;\Sigma;\Gamma_2\vdash t_2:B_1}{D;\Sigma;\Gamma_1+\Gamma_2\vdash t_1\ t_2:B_2}\ (\textsc{app})$$

最後の導出規則が (APP) であったとき、以下が成立する。

- $t=t_1\ t_2$
- $B=B_2$
- $(\Gamma,x:[A]_r,\Gamma')=(\Gamma_1+\Gamma_2)$

ここで 3 つ目の等式に注目すると、項変数環境の結合 + の定義から、項変数 x に関する仮定は項変数環境 Γ_1,Γ_2 のいずれか一方、もしくは両方に含まれる。

- $(x:[A]_r)\in\Gamma_1$ かつ $x\notin\mathrm{dom}(\Gamma_2)$ の場合
  改めて $\Gamma_1=(\Gamma_1',x:[A]_r,\Gamma_1'')$ とおく。これを用い、最後の型導出は以下のように書き換えられる。

$$\frac{D;\Sigma;\Gamma_1',x:[A]_r,\Gamma_1''\vdash t_1:B_1\to B_2 \qquad D;\Sigma;\Gamma_2\vdash t_2:B_1}{D;\Sigma;(\Gamma_1',x:[A]_r,\Gamma_1'')+\Gamma_2\vdash t_1\ t_2:B_2}\ (\textsc{app})$$

  ここで補題と直上の導出で項変数環境に注目すると

$$\begin{aligned}(\Gamma,x:[A]_r,\Gamma')&=(\Gamma_1+\Gamma_2)\\&=(\Gamma_1',x:[A]_r,\Gamma_1'')+\Gamma_2\\&=(\Gamma_1'+\Gamma_{2|\Gamma_1'}),x:[A]_r,(\Gamma_1''+\Gamma_{2|\overline{\Gamma_1'}})\end{aligned}\qquad(\because \text{A.2.5 (1)})$$

  最左辺と最右辺に注目すれば $\Gamma=(\Gamma_1'+\Gamma_{2|\Gamma_1'})$ と $\Gamma'=(\Gamma_1''+\Gamma_{2|\overline{\Gamma_1'}})$ を得られる。
  次に帰納法の仮定を 2 つの前提それぞれに適用し、改めて (APP) を適用することで以下を得る。

$$\frac{D;\Sigma;\Gamma_1'+r\cdot\Delta+\Gamma_1''\vdash[t/x]t_1:B_1\to B_2 \qquad D;\Sigma;\Gamma_2\vdash[t/x]t_2:B_1}{D;\Sigma;(\Gamma_1'+r\cdot\Delta+\Gamma_1'')+\Gamma_2\vdash([t/x]t_1)\ ([t/x]t_2):B_2}\ (\textsc{app})$$

  ここで $([t/x]t_1)\ ([t/x]t_2)=[t/x](t_1\ t_2)$ に注意すれば、直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後にこの項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma+r\cdot\Delta+\Gamma')=$

$((\Gamma_1 + r \cdot \Delta + \Gamma_1'') + \Gamma_2)$ を示す。$\Gamma = (\Gamma_1' + \Gamma_{2|\Gamma_1'})$ と $\Gamma' = (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}})$ から

$$\begin{aligned}
(\Gamma + \Delta + \Gamma') &= (\Gamma_1' + \Gamma_{2|\Gamma_1'}) + r \cdot \Delta + (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}}) \\
&= (\Gamma_1' + r \cdot \Delta + \Gamma_1'' + (\Gamma_{2|\Gamma_1'} + \Gamma_{2|\overline{\Gamma_1'}})) && (\because + \text{の可換性と結合性}) \\
&= (\Gamma_1' + r \cdot \Delta + \Gamma_1'' + \Gamma_2) && (\because \text{A.2.4}) \\
&= ((\Gamma_1' + r \cdot \Delta + \Gamma_1'') + \Gamma_2) && (\because + \text{の結合性})
\end{aligned}$$

以上より補題の結論を得る。

- $x \notin \mathrm{dom}(\Gamma_1)$ かつ $(x : [A]_r) \in \Gamma_2$ の場合
  改めて $\Gamma_2 = (\Gamma_2', x : [A]_{r_2}, \Gamma_2'')$ とおく。これを用い、最後の型導出は以下のように書き換えられる。

$$\frac{D; \Sigma; \Gamma_1 \vdash t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2', x : [A]_r, \Gamma_2'' \vdash t_2 : B_1}{D; \Sigma; \Gamma_1 + (\Gamma_2', x : [A]_r, \Gamma_2'') \vdash t_1\ t_2 : B_2}\ (\textsc{app})$$

  以下の証明は $(x : [A]_r) \in \Gamma_1$ かつ $x \notin \mathrm{dom}(\Gamma_2)$ のケースと全く同様である。ただし、項変数環境の等式変形に A.2.5 (1) ではなく A.2.5 (2) を用いる。

- $(x : [A]_{r_1}) \in \Gamma_1$ かつ $(x : [A]_{r_2}) \in \Gamma_2$ かつ $r = r_1 + r_2$ の場合
  改めて $\Gamma_1 = (\Gamma_1', x : [A]_{r_1}, \Gamma_1''), \Gamma_2 = (\Gamma_2', x : [A]_{r_2}, \Gamma_2'')$ とおく。これらを用い、最後の型導出は以下のように書き換えられる。

$$\frac{D; \Sigma; \Gamma_1', x : [A]_{r_1}, \Gamma_1'' \vdash t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2', x : [A]_{r_2}, \Gamma_2'' \vdash t_2 : B_1}{D; \Sigma; (\Gamma_1', x : [A]_{r_1}, \Gamma_1'') + (\Gamma_2', x : [A]_{r_2}, \Gamma_2'') \vdash t_1\ t_2 : B_2}\ (\textsc{app})$$

  ここで補題と直上の導出で項変数環境に注目すると

$$\begin{aligned}
(\Gamma, x : [A]_r, \Gamma') &= (\Gamma_1 + \Gamma_2) \\
&= (\Gamma_1', x : [A]_{r_1}, \Gamma_1'') + (\Gamma_2', x : [A]_{r_2}, \Gamma_2'') \\
&= (\Gamma_1', \Gamma_1'', x : [A]_{r_1}) + (\Gamma_2', \Gamma_2'', x : [A]_{r_2}) && (\because \text{, の可換性}) \\
&= ((\Gamma_1', \Gamma_1'') + (\Gamma_2', \Gamma_2'')), x : [A]_{r_1 \oplus r_2} && (\because + \text{の定義}) \\
&= ((\Gamma_1 + \Gamma_{2|\Gamma_1'}' + \Gamma_{2|\Gamma_1'}''), (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}}' + \Gamma_{2|\overline{\Gamma_1'}}')), x : [A]_{r_1 \oplus r_2} && (\because \text{A.2.5 (3)}) \\
&= ((\Gamma_1 + \Gamma_{2|\Gamma_1'}' + \Gamma_{2|\Gamma_1'}''), x : [A]_{r_1 \oplus r_2}, (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}}' + \Gamma_{2|\overline{\Gamma_1'}}')) && (\because \text{, の可換性})
\end{aligned}$$

  最左辺と最右辺に注目すれば $\Gamma = (\Gamma_1 + \Gamma_{2|\Gamma_1'}' + \Gamma_{2|\Gamma_1'}'')$ と $\Gamma' = (\Gamma_1'' + \Gamma_{2|\overline{\Gamma_1'}}' + \Gamma_{2|\overline{\Gamma_1'}}')$ を得られる。
  次に帰納法の仮定を 2 つの前提それぞれに適用し、改めて (app) を適用することで以下を得る。

$$\frac{D; \Sigma; \Gamma_1' + r_1 \cdot \Delta + \Gamma_1'' \vdash [t/x]t_1 : B_1 \to B_2 \qquad D; \Sigma; \Gamma_2' + r_2 \cdot \Delta + \Gamma_2'' \vdash [t/x]t_2 : B_1}{D; \Sigma; (\Gamma_1' + r_1 \cdot \Delta + \Gamma_1'') + (\Gamma_2' + r_2 \cdot \Delta + \Gamma_2'') \vdash ([t/x]t_1)\ ([t/x]t_2) : B_2}\ (\textsc{app})$$

  ここで $([t/x]t_1)\ ([t/x]t_2) = [t/x](t_1\ t_2)$ に注意すれば、直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後にこの項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma_1' + r_1 \cdot \Delta + \Gamma_1'') + (\Gamma_2' +$

$r_2 \cdot \Delta + \Gamma_2'') = \Gamma + r \cdot \Delta + \Gamma'$ を示す。$\Gamma = (\Gamma_1 + \Gamma_2'_{|\Gamma_1'} + \Gamma_2''_{|\Gamma_1'})$ と $\Gamma' = (\Gamma_1'' + \Gamma_2'_{|\overline{\Gamma_1'}} + \Gamma_2'_{|\overline{\Gamma_1'}})$ から

$$\begin{aligned}
(\Gamma + r \cdot \Delta + \Gamma') &= (\Gamma_1 + \Gamma_2'_{|\Gamma_1'} + \Gamma_2''_{|\Gamma_1'}) + (r_1 \oplus r_2) \cdot \Delta + (\Gamma_1'' + \Gamma_2'_{|\overline{\Gamma_1'}} + \Gamma_2'_{|\overline{\Gamma_1'}}) \\
&= \Gamma_1 + (r_1 \oplus r_2) \cdot \Delta + \Gamma_1'' + (\Gamma_2'_{|\Gamma_1'} + \Gamma_2'_{|\overline{\Gamma_1'}}) + (\Gamma_2''_{|\Gamma_1'} + \Gamma_2'_{|\overline{\Gamma_1'}}) && (\because + \text{の可換性と結合性}) \\
&= \Gamma_1' + (r_1 \oplus r_2) \cdot \Delta + \Gamma_1'' + \Gamma_2' + \Gamma_2'' && (\because \text{A.2.4}) \\
&= \Gamma_1' + r_1 \cdot \Delta + r_2 \cdot \Delta + \Gamma_1'' + \Gamma_2' + \Gamma_2'' && (\because \text{A.2.6}) \\
&= (\Gamma_1' + r_1 \cdot \Delta + \Gamma_1'') + (\Gamma_2' + r_2 \cdot \Delta + \Gamma_2'') && (\because + \text{の可換性と結合性})
\end{aligned}$$

以上より補題の結論を得る。

$$\text{Case. } \frac{}{D; \Delta; \emptyset \vdash n : \mathsf{Int}}\ (\textsc{int}) \quad \& \quad \text{Case. } \frac{(x : \forall\{\overrightarrow{\alpha : \kappa}\}.A) \in D \quad \theta, \Sigma' = \mathsf{inst}(\overrightarrow{\alpha : \kappa})}{D; \Sigma, \Sigma'; \emptyset \vdash x : \theta A}\ (\mathrm{C})$$

(INT), (C) の結論の項変数環境は空（$= \emptyset$）である。従って補題の結論は自明に成立する。

$$\text{Case. } \frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma'' \vdash t : B}{D; \Sigma; \Gamma'' + [\Delta']_{0:R} \vdash t : B}\ (\textsc{weak})$$

最後の導出規則が (WEAK) であったとき、補題と (WEAK) で項変数環境に注目すれば $(\Gamma, x : [A]_r, \Gamma') = \Gamma'' + [\Delta']_{0:R}$ である。このときコエフェクト仮定 $x : [A]_r$ はコエフェクト付き項変数環境 $[\Delta']_{0:R}$ に含まれる場合と含まない場合がある。

- $(x : [A]_r) \in [\Delta']_{0:R}$ の場合

  $r = 0$ であり、改めて $\Delta' = (\Delta_1, x : [A]_0, \Delta_2)$ とおく。このとき $x : [A]_r \notin \Gamma''$ としてよい。ここで補題と直上の導出で項変数環境に注目すると

$$\begin{aligned}
(\Gamma, x : [A]_0, \Gamma') &= \Gamma'' + ([\Delta_1]_{0:R}, x : [A]_0, [\Delta_2]_{0:R}) \\
&= (\Gamma''_{|[\Delta_2]_{0:R}} + [\Delta_1]_{0:R}), x : [A]_0, (\Gamma''_{|\overline{[\Delta_2]_{0:R}}} + [\Delta_2]_{0:R}) && (\because \text{A.2.5 (2)})
\end{aligned}$$

  最左辺と最右辺に注目すれば $\Gamma = (\Gamma''_{|[\Delta_2]_{0:R}} + [\Delta_1]_{0:R})$ と $\Gamma' = (\Gamma''_{|\overline{[\Delta_2]_{0:R}}} + [\Delta_2]_{0:R})$ である。

  帰納法の仮定から前提に補題を適用し、再び (WEAK) を適用することで以下を得る。このとき弱化で追加される環境として、新しく $\Delta_1 + \Delta + \Delta_2$ を選択する。

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma'' \vdash [t'/x]t : B}{D; \Sigma; \Gamma'' + [\Delta_1 + \Delta + \Delta_2]_{0:R} \vdash [t'/x]t : B}\ (\textsc{weak})$$

  ここで x は項 t の中で使われていないため、$[t'/x]t = t$ であることに注意すれば、直上の導出の結論は項変数環境を除いて補題の結論と等しい。

  最後にこの項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma + r \cdot \Delta + \Gamma') = \Gamma'' + [\Delta_1 + \Delta + \Delta_2]_{0:R}$

を示す。$r = 0$ と $\Gamma = (\Gamma''_{|[\Delta_2]_{0:R}} + [\Delta_1]_{0:R})$ と $\Gamma' = (\Gamma''_{|\overline{[\Delta_2]_{0:R}}} + [\Delta_2]_{0:R})$ から

$$\begin{aligned}(\Gamma + r \cdot \Delta + \Gamma') &= (\Gamma''_{|[\Delta_2]_{0:R}} + [\Delta_1]_{0:R}) + [\Delta]_{0:R} + (\Gamma''_{|\overline{[\Delta_2]_{0:R}}} + [\Delta_2]_{0:R}) \\ &= (\Gamma''_{|[\Delta_2]_{0:R}} + \Gamma''_{|\overline{[\Delta_2]_{0:R}}}) + ([\Delta_1]_{0:R} + [\Delta]_{0:R} + [\Delta_2]_{0:R}) && (\because + \text{の結合性と可換性}) \\ &= \Gamma'' + [\Delta_1 + \Delta + \Delta_2]_{0:R} && (\because \text{A.2.4})\end{aligned}$$

以上より補題の結論を得る。

- $(x : [A]_r) \notin [\Delta']_{0:R}$ の場合

改めて $\Gamma'' = (\Gamma_1, x : [A]_r, \Gamma_2)$ とおく。これを用い、最後の型導出は以下のように書き換えられる。

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; (\Gamma_1, x : [A]_r, \Gamma_2) \vdash t : B}{D; \Sigma; (\Gamma_1, x : [A]_r, \Gamma_2) + [\Delta']_{0:R} \vdash t : B}\ (\textsc{weak})$$

ここで補題と (weak) で項変数環境に注目すると

$$\begin{aligned}(\Gamma, x : [A]_r, \Gamma') &= (\Gamma_1, x : [A]_r, \Gamma_2) + [\Delta']_{0:R} \\ &= (\Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1}), x : [A]_r, (\Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}}) && (\because \text{A.2.5 (1)})\end{aligned}$$

最左辺と最右辺に注目すれば、$\Gamma = (\Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1})$ と $\Gamma' = (\Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}})$ を得る。また帰納法の仮定から補題を前提に適用し、改めて (weak) を適用することで以下を得る。

$$\frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma_1 + r \cdot \Delta + \Gamma_2 \vdash [t'/x]t : B}{D; \Sigma; (\Gamma_1 + r \cdot \Delta + \Gamma_2) + [\Delta']_{0:R} \vdash [t'/x]t : B}\ (\textsc{weak})$$

直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後に、この項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma + r \cdot \Delta + \Gamma') = (\Gamma_1 + r \cdot \Delta + \Gamma_2) + [\Delta']_{0:R}$ を示す。$\Gamma = (\Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1})$ と $\Gamma' = (\Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}})$ から

$$\begin{aligned}(\Gamma + r \cdot \Delta + \Gamma') &= (\Gamma_1 + ([\Delta']_{0:R})_{|\Gamma_1}) + r \cdot \Delta + (\Gamma_2 + ([\Delta']_{0:R})_{|\overline{\Gamma_1}}) \\ &= \Gamma_1 + r \cdot \Delta + \Gamma_2 + ([\Delta']_{0:R\,|\Gamma_1} + [\Delta']_{0:R\,|\overline{\Gamma_1}}) && (\because + \text{の結合性と可換性}) \\ &= \Gamma_1 + r \cdot \Delta + \Gamma_2 + [\Delta']_{0:R} && (\because \text{A.2.4}) \\ &= (\Gamma_1 + r \cdot \Delta + \Gamma_2) + [\Delta']_{0:R} && (\because + \text{の結合性})\end{aligned}$$

以上より補題の結論を得る。

Case. $$\frac{\Sigma \vdash R : \mathsf{Coeff} \qquad D; \Sigma; \Gamma'', y : B_1 \vdash t : B_2}{D; \Sigma; \Gamma'', y : [B_1]_{1:R} \vdash t : B_2}\ (\textsc{der})$$

最後の導出規則が (der) であったとき、補題と (der) で項変数環境に注目すれば $(\Gamma'', y : [B_1]_{1:R}) = (\Gamma, x : [A]_r, \Gamma')$ である。このときコエフェクト仮定 $x : [A]_r$ が $y : [B_1]_{1:R}$ と等しい場合と等しくない場合それぞれを考える。

- $x : [A]_r = y : [B_1]_{1:R}$ の場合

$x = y, A = B_1, r = 1, \Gamma = \Gamma'', \Gamma' = \emptyset$ であり、これらを用いて最後の導出は以下のように書き換えられる。

$$\frac{\Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; \Gamma'', x : A \vdash t : B_2}{D; \Sigma; \Gamma'', x : [A]_{1:R} \vdash t : B_2}\ (\textsc{der})$$

前提に A.3.1 を適用し

$$D; \Sigma; \Gamma'' + \Delta \vdash [t'/x]t : B_2$$

を得る。A.2.2 より $\Gamma'' + \Delta = \Gamma'' + r \cdot \Delta$ に注意すれば、これは補題の結論そのものである。

- $x : [A]_r \neq y : [B_1]_{1:R}$ の場合
  改めて $\Gamma'' = (\Gamma_1, x : [A]_r, \Gamma_1')$ とおく。これを用い、最後の導出は以下のように書き換えられる。

$$\frac{\Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; (\Gamma_1, x : [A]_r, \Gamma_1'), y : B_1 \vdash t : B_2}{D; \Sigma; (\Gamma_1, x : [A]_r, \Gamma_1'), y : [B_1]_{1:R} \vdash t : B_2}\ (\textsc{der})$$

  帰納法の仮定から補題を前提に適用し、改めて (der) を適用することで以下を得る。

$$\frac{\Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; (\Gamma + r \cdot \Delta + \Gamma''), y : B_1 \vdash [t'/x]t : B_2}{D; \Sigma; (\Gamma + r \cdot \Delta + \Gamma''), y : [B_1]_{1:R} \vdash [t'/x]t : B_2}\ (\textsc{der})$$

  最後に、$y : [B_1]_{1:R}$ は $\Gamma + r \cdot \Delta + \Gamma''$ と非交であるから、$((\Gamma + r \cdot \Delta + \Gamma''), y : [B_1]_{1:R}) = \Gamma + r \cdot \Delta + (\Gamma'', y : [B_1]_{1:R})$ なことに注意すれば、直上の導出式は補題の結論そのものである。

$$\text{Case.}\quad \frac{\Sigma \vdash r' : \uparrow R \quad \Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; [\Gamma_1] \vdash t : B}{D; \Sigma; r' \cdot [\Gamma_1] \vdash [t] : \square_{r'} B}\ (\textsc{pr})$$

最後の導出規則が (pr) であったとき、ある $r'' \sqsubseteq r'$ が存在して $[\Gamma_1] = [\Gamma_1', x : [A]_{r''}, \Gamma_1'']$ と表せる。これを用いて最後の導出を書き直すと

$$\frac{\Sigma \vdash r' : \uparrow R \quad \Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; [\Gamma_1', x : [A]_{r''}, \Gamma_1''] \vdash t : B}{D; \Sigma; r' \cdot [\Gamma_1', x : [A]_{r''}, \Gamma_1''] \vdash [t] : \square_{r'} B}\ (\textsc{pr})$$

補題と導出の結論の項変数環境に注目すると、

$$\begin{aligned}
(\Gamma, x : [A]_r, \Gamma') &= r' \cdot [\Gamma_1] \\
&= r' \cdot [\Gamma_1', x : [A]_{r''}, \Gamma_1''] \\
&= r' \cdot [\Gamma_1'],\ x : [A]_{r'' \otimes r'},\ r' \cdot [\Gamma_1''] && (\because \cdot \text{ の分配則}) \\
&= r' \cdot [\Gamma_1'],\ x : [A]_{r'},\ r' \cdot [\Gamma_1''] && (\because r'' \sqsubseteq r')
\end{aligned}$$

最左辺と最右辺に注目すれば、$\Gamma = (r' \cdot [\Gamma_1'])$ と $\Gamma' = (r' \cdot [\Gamma_1''])$ を得る。また帰納法の仮定から補題を前提に適用し、改めて (pr) を適用することで以下を得る。

$$\frac{\Sigma \vdash r' : \uparrow R \quad \Sigma \vdash R : \mathrm{Coeff} \qquad D;\Sigma;[\Gamma_1' + r'' \cdot \Delta + \Gamma_1''] \vdash [t'/x]t : B}{D;\Sigma;r' \cdot [\Gamma_1' + r'' \cdot \Delta + \Gamma_1''] \vdash [[t'/x]t] : \Box_{r'} B}\ (\textsc{pr})$$

$[[t'/x]t] = [t'/x][t]$ に注意すれば、直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後に、この項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma + r' \cdot \Delta + \Gamma') = r' \cdot [\Gamma_1' + r'' \cdot \Delta + \Gamma_1'']$ を示す。$\Gamma = (r' \cdot [\Gamma_1'])$ と $\Gamma' = (r' \cdot [\Gamma_1''])$ から

$$\begin{aligned}
(\Gamma + r' \cdot \Delta + \Gamma') &= r' \cdot [\Gamma_1'] + r' \cdot \Delta + r' \cdot [\Gamma_1'] \\
&= r' \cdot [\Gamma_1'] + (r' \otimes r'') \cdot \Delta + r' \cdot [\Gamma_1'] && (\because r'' \sqsubseteq r') \\
&= r' \cdot [\Gamma_1'] + r' \cdot (r'' \cdot \Delta) + r' \cdot [\Gamma_1'] && (\because \otimes \text{の結合性}) \\
&= r' \cdot [\Gamma_1' + r'' \cdot \Delta + \Gamma_1'] && (\because \cdot \text{の分配則})
\end{aligned}$$

以上より補題の結論を得る。

Case. $$\frac{D;\Sigma;[\Gamma_i] \vdash t_i : B}{D;\Sigma;\bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \{\overline{l = t} | l_i\} : \Box_{\overline{l}} B}\ (\textsc{ver})$$

最後の導出規則が (ver) のとき、まず以下のマップ関数 σ_i を定義する。

$$\sigma_i = \begin{cases} 1 & (x \in \mathrm{dom}(\Gamma_i)) \\ 0 & (x \notin \mathrm{dom}(\Gamma_i)) \end{cases} \qquad \sum_i (\sigma_i \otimes \{l_i\}) = \{l_i \mid x \in \mathrm{dom}(\Gamma_i)\} = r$$

σ_i を用いることで以下の導出を得る。

- $\sigma_i = 1$、すなわち $x \in \mathrm{dom}(\Gamma_i)$ の場合

  $$\begin{aligned}
  [\Gamma_i] &= [\Gamma_i', x : [A]_1, \Gamma_i''] \\
  &= [\Gamma_i', x : [A]_{\sigma_i}, \Gamma_i'']
  \end{aligned}$$

  と表すことが出来る。
- $\sigma_i = 0$、すなわち $x \notin \mathrm{dom}(\Gamma_i)$ の場合

  (weak) を用いると

  $$\frac{\Sigma \vdash R : \mathrm{Coeff} \quad D;\Sigma;[\Gamma_i] \vdash t_i : B}{D;\Sigma;[\Gamma_i] + (x : [A]_{0:R}) \vdash t_i : B}\ (\textsc{weak})$$

  直上の導出の項変数環境に注目すると、適当な Γ_i の非交部分列 Γ_i', Γ_i'' で以下が成立する。

  $$\begin{aligned}
  [\Gamma_i] + (x : [A]_{0:R}) &= [\Gamma_i', x : [A]_{0:R}, \Gamma_i''] \\
  &= [\Gamma_i', x : [A]_{\sigma_i}, \Gamma_i'']
  \end{aligned}$$

以上を踏まえると、任意の i について適当な非交部分列 Γ_i', Γ_i'' を選ぶことで

$$[\Gamma_i] = [\Gamma_i', x : [A]_{\sigma_i}, \Gamma_i'']$$

なる分割が成立する。

この分割を用いて最後の導出を書き直すと以下のようになる。

$$\frac{D;\Sigma;[\Gamma_i', x:[A]_{\sigma_i}, \Gamma_i''] \vdash t_i : B}{D;\Sigma;\bigcup_i(\{l_i\}\cdot[\Gamma_i', x:[A]_{\sigma_i}, \Gamma_i'']) \vdash \{\overline{l=t}|l_i\} : \Box_{\overline{l}}B}\ (\textsc{ver})$$

ここで最後の導出と補題の結論の項変数環境に注目すると

$$\begin{aligned}
(\Gamma, x:[A]_r, \Gamma') &= \bigcup_i (\{l_i\}\cdot[\Gamma_i]) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i',\ x:[A]_{\sigma_i},\ \Gamma_i'']\right) && (\because [\Gamma_i] = [\Gamma_i', x:[A]_{\sigma_i}, \Gamma_i'']) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'],\ \{l_i\}\cdot(x:[A]_{\sigma_i}),\ \{l_i\}\cdot[\Gamma_i'']\right) && (\because \cdot \text{ の分配則}) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right), \bigcup_i \left(\{l_i\}\cdot(x:[A]_{\sigma_i})\right), \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) && (\because \text{各非交部分列の和を計算}) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right), \bigcup_i \left(x:[A]_{\sigma_i \otimes \{l_i\}}\right), \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) && (\because \cdot \text{ の定義}) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right),\ x:[A]_{\Sigma_i(\sigma_i \otimes \{l_i\})}, \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) && (\because \bigcup \text{ の定義}) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right),\ x:[A]_r, \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) && (\because \sigma_i \text{ の定義})
\end{aligned}$$

最左辺と最右辺に注目すれば、$\Gamma = \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right)$ と $\Gamma' = \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right)$ を得る。また帰納法の仮定から補題を前提に適用し、改めて (ver) を適用することで以下を得る。

$$\frac{D;\Sigma;[\Gamma_i' + \sigma_i\cdot\Delta + \Gamma_i''] \vdash [t'/x]t_i : B}{D;\Sigma;\bigcup_i(\{l_i\}\cdot[\Gamma_i' + \sigma_i\cdot\Delta + \Gamma_i'']) \vdash \{\overline{l=[t'/x]t}|l_i\} : \Box_{\overline{l}}B}\ (\textsc{ver})$$

$\{\overline{l=[t'/x]t}|l_i\} = [t'/x]\{\overline{l=t}|l_i\}$ に注意すれば、直上の導出の結論は項変数環境を除いて補題の結論と等しい。最後に、この項変数環境が補題の結論のものに等しいこと、すなわち $(\Gamma + r\cdot\Delta + \Gamma') = \bigcup_i(\{l_i\}\cdot[\Gamma_i' + \sigma_i\cdot\Delta + \Gamma_i''])$ を示す。$\Gamma = \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right)$ と $\Gamma' = \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right)$ から

$$\begin{aligned}
(\Gamma + r\cdot\Delta + \Gamma') &= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right) + r\cdot\Delta + \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right) + \left(\sum_i (\sigma_i \otimes \{l_i\})\right)\cdot\Delta + \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i']\right) + \bigcup_i (\{l_i\}\cdot(\sigma_i\cdot\Delta)) + \bigcup_i \left(\{l_i\}\cdot[\Gamma_i'']\right) \\
&= \bigcup_i \left(\{l_i\}\cdot\left([\Gamma_i'] + (\sigma_i\cdot\Delta) + [\Gamma_i'']\right)\right) \\
&= \bigcup_i \left(\{l_i\}\cdot[\Gamma_i' + \sigma_i\cdot\Delta + \Gamma_i'']\right)
\end{aligned}$$

以上より補題の結論を得る。

$$\text{Case. } \frac{D;\Sigma;[\Gamma_i] \vdash t_i : B}{D;\Sigma;\bigcup_i(\{l_i\}\cdot[\Gamma_i]) \vdash \langle\overline{l=t}|l_i\rangle : B}\ (\textsc{veri})$$

(ver) と全く同様。

$$\text{Case.}\ \frac{D;\Sigma;\Gamma \vdash t : \square_r A \quad l \in r}{D;\Sigma;\Gamma \vdash t.l : A}\ (\textsc{rstr})$$

最後の導出規則が (RSTR) であったとき、帰納法の仮定から前提に補題を適用し、得られた導出式に改めて (RSTR) を適用することで直接補題の結論を得る。

$$\text{Case.}\quad \frac{D;\Sigma;\Gamma_1, y : [B']_{r_1}, \Gamma_2 \vdash t : B \quad r_1 \sqsubseteq r_2}{D;\Sigma;\Gamma_1, y : [B']_{r_2}, \Gamma_2 \vdash t : B}\ (\sqsubseteq)$$

最後の導出規則が (⊑) であったとき、補題と (⊑) で項変数環境に注目すれば $(\Gamma, x : [A]_r, \Gamma') = (\Gamma, y : [B']_{r_2}, \Gamma')$ である。このときコエフェクト仮定 $x : [A]_r$ は Γ_1 に含まれる場合, Γ_2 に含まれる場合, $y : [B']_{r_2}$ と等しい場合の 3 パターン存在する。

- $(x : [A]_r) \in \Gamma_1$ の場合
  改めて $\Gamma_1 = (\Gamma_1', x : [A]_r, \Gamma_1'')$ とおく。これを用いて最後の導出を書き直すと
  $$\frac{D;\Sigma;\Gamma_1', x : [A]_r, \Gamma_1'', y : [B']_{r_1}, \Gamma_2 \vdash t : B \quad r_1 \sqsubseteq r_2}{D;\Sigma;\Gamma_1', x : [A]_r, \Gamma_1'', y : [B']_{r_2}, \Gamma_2 \vdash t : B}\ (\sqsubseteq)$$
  帰納法の仮定から前提に補題を適用でき、以下を得る。
  $$D;\Sigma;\Gamma_1' + r \cdot \Delta + (\Gamma_1'', y : [B']_{r_1}, \Gamma_2) \vdash [t'/x]t : B \tag{A.1}$$
  直上に導出式の項変数環境に注目すれば
  $$\begin{aligned}
  &\Gamma_1' + r \cdot \Delta + (\Gamma_1'', y : [B']_{r_1}, \Gamma_2) \\
  &= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_1}, \Gamma_2))}}, \\
  &\qquad \left((\Gamma_1'', y : [B']_{r_1}, \Gamma_2) + (r \cdot \Delta)_{|(\Gamma_1'', y:[B']_{r_1}, \Gamma_2)}\right) && (\because \text{A.2.7}) \\
  &= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_1}, \Gamma_2))}}, \\
  &\qquad \left((\Gamma_1'', y : [B']_{r_1}, \Gamma_2) + \left((r \cdot \Delta)_{|\Gamma_1''}, (r \cdot \Delta)_{|(y:[B']_{r_1})}, (r \cdot \Delta)_{|\Gamma_2}\right)\right) && (\because \text{A.2.3}) \\
  &= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_1}, \Gamma_2))}}, \\
  &\qquad \left(\Gamma_1'' + (r \cdot \Delta)_{|\Gamma_1''}\right), \left(y : [B']_{r_1} + (r \cdot \Delta)_{|(y:[B']_{r_1})}\right), \left(\Gamma_2 + (r \cdot \Delta)_{|\Gamma_2}\right) && (\because \text{A.2.5}) \\
  &= \Gamma_3, y : [B']_{r_1 \oplus r_3}, \Gamma_3'
  \end{aligned}$$
  最後の変形は
  $$\begin{aligned}
  \Gamma_3 &= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_1}, \Gamma_2))}}, \left(\Gamma_1'' + (r \cdot \Delta)_{|\Gamma_1''}\right) \\
  \Gamma_3' &= \left(\Gamma_2 + (r \cdot \Delta)_{|\Gamma_2}\right)
  \end{aligned}$$
  と置いた上で、以下の等式 A.2 を踏まえたものである。$(r \cdot \Delta)_{|(y:[B']_{r_1})}$ に注目すると、適当な r_3' と $r_3 = r \otimes r_3'$ が存在して
  $$(r \cdot \Delta)_{|(y:[B']_{r_1})} = \begin{cases} r \cdot (y : [B']_{r_3'}) = y : [B']_{r \otimes r_3'} = y : [B']_{r_3} & (y \in \mathrm{dom}(\Delta)) \\ \emptyset & (y \notin \mathrm{dom}(\Delta)) \end{cases}$$

この等式を踏まえると以下を得る。

$$y : [B']_{r_1} + (r \cdot \Delta)_{|(y:[B']_{r_1})} = \begin{cases} y : [B']_{r_1 \oplus r_3} & (y \in \mathrm{dom}(\Delta)) \\ y : [B']_{r_1 \oplus r_3} = y : [B']_{r_1} & (y \notin \mathrm{dom}(\Delta)) \end{cases} \tag{A.2}$$

以上の変形を全て適用し、式 A.1 に改めて (⊑) を適用して以下を得る。

$$\frac{D; \Sigma; \Gamma_3, y : [B']_{r_1 \oplus r_3}, \Gamma_3' \vdash [t'/x]t : B \quad (r_1 \oplus r_3) \sqsubseteq (r_2 \oplus r_3)}{D; \Sigma; \Gamma_3, y : [B']_{r_2 \oplus r_3}, \Gamma_3' \vdash [t'/x]t : B} \ (\sqsubseteq)$$

直上の導出の結論は項変数環境を除いて補題の結論と等しい。

最後にこの項変数環境が補題の結論のものに等しいこと、すなわち $\Gamma_1' + r \cdot \Delta + (\Gamma_1'', y : [B']_{r_2}, \Gamma_2) = (\Gamma_3, y : [B']_{r_2 \oplus r_3}, \Gamma_3')$ を示す。

$$\begin{aligned}
&\Gamma_1' + r \cdot \Delta + (\Gamma_1'', y : [B']_{r_2}, \Gamma_2) \\
&= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_2}, \Gamma_2))}}, \\
&\qquad \left((\Gamma_1'', y : [B']_{r_2}, \Gamma_2) + (r \cdot \Delta)_{|(\Gamma_1'', y:[B']_{r_2}, \Gamma_2)}\right) && (\because \text{A.2.7}) \\
&= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_2}, \Gamma_2))}}, \\
&\qquad \left((\Gamma_1'', y : [B']_{r_2}, \Gamma_2) + \left((r \cdot \Delta)_{|\Gamma_1''}, (r \cdot \Delta)_{|(y:[B']_{r_2})}, (r \cdot \Delta)_{|\Gamma_2}\right)\right) && (\because \text{A.2.3}) \\
&= (\Gamma_1' + (r \cdot \Delta)_{|\Gamma_1'}), (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_2}, \Gamma_2))}}, \\
&\qquad \left(\Gamma_1'' + (r \cdot \Delta)_{|\Gamma_1''}\right), \left(y : [B']_{r_2} + (r \cdot \Delta)_{|(y:[B']_{r_2})}\right), \left(\Gamma_2 + (r \cdot \Delta)_{|\Gamma_2}\right) && (\because \text{A.2.5}) \\
&= \Gamma_3, y : [B']_{r_2 \oplus r_3}, \Gamma_3'
\end{aligned}$$

最後の変形は定義 A.2.3 から導出可能な以下の等式による。

$$\begin{aligned}
(r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_1}, \Gamma_2))}} &= (r \cdot \Delta)_{|\overline{(\Gamma_1', (\Gamma_1'', y:[B']_{r_2}, \Gamma_2))}} \\
(r \cdot \Delta)_{|(y:[B']_{r_1})} &= (r \cdot \Delta)_{|(y:[B']_{r_2})}
\end{aligned}$$

以上より補題の結論を得る。

- $(x : [A]_r) \in \Gamma_2$ の場合

  $(x : [A]_r) \in \Gamma_1$ の場合と全く同様である。
- $(x : [A]_r) = y : [B']_{r_2}$ の場合

  最後の型導出は以下のように書き換えられる。

$$\frac{D; \Sigma; \Gamma_1, x : [A]_{r'}, \Gamma_2 \vdash t : B \quad r' \sqsubseteq r}{D; \Sigma; \Gamma_1, x : [A]_r, \Gamma_2 \vdash t : B} \ (\sqsubseteq)$$

  帰納の仮定より前提に補題を適用でき、改めて (⊑) を適用することで直接補題の結論を得る。

□

A.4 型安全性

補題 A.4.1 [逆転補題]

値 v が $D; \Sigma; \Gamma \vdash v : A$ を満たすとき、各型 A について以下が成立する。

- $A = \mathsf{Int} \Longrightarrow$ ある整数定数 n が存在して $v = n$

- $A = \square_r B \Longrightarrow$ ある項 t' が存在して $v = [t']$、もしくは項 t_i が存在して $v = \{\overline{l = t} \mid l_i \in r\}$
- $A = B \to B' \Longrightarrow$ あるパターン p と項 t が存在して、$v = \lambda p.t$
- otherwise $\Longrightarrow \Gamma$ 中のある変数 x について $v = x$

補題 A.4.2 [デフォルトバージョン上書き @ に関する型安全性]

バージョンタグ $l \in r$ と項 t についてある t' が存在し、以下が成立する。

$$\left.\begin{array}{r} D;\Sigma;[\Gamma'] \vdash t : A \\ l \in r \end{array}\right\} \quad \Longrightarrow \quad \exists t'. \begin{cases} t@l = t' & \text{（進行性）} \\ D;\Sigma;\{l\} \cdot [\Gamma'] \vdash t' : A & \text{（保存性）} \end{cases}$$

証明

$D;\Sigma;[\Gamma'] \vdash t : A$ の型導出に関する構造的帰納法による。 □

補題 A.4.3 [線形パターンに関する型安全性]

パターン p と項 t, t'' についてある t' が存在し、以下が成立する。

$$\left.\begin{array}{r} D;\Sigma;- \vdash p : A \rhd \Delta \\ D;\Sigma;\Gamma_2 \vdash t'' : A \\ D;\Sigma;\Gamma_1, \Delta \vdash t : B \end{array}\right\} \quad \Longrightarrow \quad \exists t'. \begin{cases} (t'' \rhd p)t = t' & \text{（進行性）} \\ D;\Sigma;\Gamma_1 + \Gamma_2 \vdash t' : B & \text{（保存性）} \end{cases}$$

証明

$D;\Sigma;- \vdash p : A \rhd \Delta$ の型導出に関する構造的帰納法で示す。最後に用いたパターン型導出規則で場合分け。

$$\text{Case.} \quad \frac{\Sigma \vdash A : \mathsf{Type}}{D;\Sigma;- \vdash x : A \ \rhd \ x : A}\ (\textsc{pVar})$$

最後のパターン型導出が (pVar) であったとき、$p = x, \Delta = x : A$ である。このとき縮約規則 $(\rhd_{\mathrm{var}})$ が適用できる。

$$\frac{}{(t'' \rhd x)t = [t''/x]t}\ (\rhd_{\mathrm{var}})$$

従って $t' = [t''/x]t$ とすれば補題の結論（進行性）を得る。 今、$\Delta = (x : A)$ を補題の仮定に代入し

$$D;\Sigma;\Gamma_1, x : A \vdash t : B$$

である。これに A.3.1 を適用することで以下を得る。

$$D;\Sigma;\Gamma_1 + \Gamma_2 \vdash [t''/x]t : B$$

これは補題の結論（保存性）そのものである。

$$\text{Case.} \quad \frac{\Sigma \vdash A : \mathsf{Type}}{D;\Sigma;r : R \vdash x : A \ \rhd \ x : [A]_{r:R}}\ ([\textsc{pVar}])$$

$r : R \neq -$ であり、仮定を満たさない。従って補題の結論は自明に成立する。

$$\text{Case.}\quad \frac{D;\Sigma;r:R\vdash x:A\ \rhd\ \Delta \qquad \Sigma\vdash r:\uparrow R}{D;\Sigma;-\vdash [x]:\Box_r A\ \rhd\ \Delta}\ (\textsc{p}\Box)$$

λ_{VL} の unbox パターンは一重のものしか出現しないので、パターン型導出木は以下のように一意に定まる。

$$\frac{D;\Sigma;r:R\vdash x:A\ \rhd\ x:[A]_r \qquad \Sigma\vdash r:\uparrow R}{D;\Sigma;-\vdash [x]:\Box_r A\ \rhd\ x:[A]_r}\ (\textsc{p}\Box)$$

今 $\Delta=(x:[A]_r)$ であるから、これを補題の仮定に代入して

$$D;\Sigma;\Gamma_1,x:[A]_r\vdash t:B$$

である。これに A.3.2 を適用することで以下を得る。

$$D;\Sigma;\Gamma_1+r\cdot\Gamma_2\vdash [t''/x]t:B$$

これは補題の結論（保存性）そのものである。

$$\text{Case.}\quad \frac{r\sqsubseteq 0 \qquad D;\Sigma\vdash A:\mathsf{Type}}{D;\Sigma;r:R\vdash \_:A\ \rhd\ \emptyset}\ (\textsc{p}\_)$$

$r:R\neq -$ であり、仮定を満たさない。従って補題の結論は自明に成立する。

□

補題 A.4.4 [コエフェクト付きパターンに関する型安全性]

パターン p と項 t,t'' についてある t' が存在し、以下が成立する。

$$\left.\begin{array}{c} D;\Sigma;r:R\vdash p:A\rhd\Delta \\ D;\Sigma;[\Gamma_2]\vdash t'':A \\ D;\Sigma;\Gamma_1,\Delta\vdash t:B \end{array}\right\} \implies \exists t'.\begin{cases} (t''\rhd p)t=t' & \text{（進行性）} \\ D;\Sigma;\Gamma_1+r\cdot\Gamma_2\vdash t':B & \text{（保存性）}\end{cases}$$

証明

$D;\Sigma;r:R\vdash p:A\rhd\Delta$ の型導出に関する構造的帰納法で示す。最後に用いたパターン型導出規則で場合分け。

$$\text{Case.}\quad \frac{\Sigma\vdash A:\mathsf{Type}}{D;\Sigma;-\vdash x:A\ \rhd\ x:A}\ (\textsc{pVar})$$

$-\neq r:R$ であり、仮定を満たさない。従って補題の結論は自明に成立する。

$$\text{Case.}\quad \frac{\Sigma\vdash A:\mathsf{Type}}{D;\Sigma;r:R\vdash x:A\ \rhd\ x:[A]_{r:R}}\ ([\textsc{pVar}])$$

最後のパターン型導出が $([\textsc{pVar}])$ の場合、$p=x,\Delta=x:[A]_{r:R}$ である。このとき $(\rhd_{\mathrm{var}})$ が適用でき、以下を得る。

$$\frac{}{(t'' \rhd x)t = [t''/x]t}\ (\rhd_{\mathrm{var}})$$

従って $t' = [t''/x]t$ とすれば補題の結論（進行性）を得る。今 $D; \Sigma; \Gamma_1, x : [A]_{r:R} \vdash t : B$ かつ $D; \Sigma; [\Gamma_2] \vdash t'' : A$ であり、A.3.2 を適用することで以下を得る。

$$D; \Sigma; \Gamma_1 + r \cdot \Gamma_2 \vdash [t''/x]t : A$$

これは補題の結論（保存性）そのものである。

$$\text{Case.}\quad \frac{\begin{array}{c} D; \Sigma; r : R \vdash x : A \ \rhd\ \Delta \\ \Sigma \vdash r : \uparrow R \end{array}}{D; \Sigma; - \vdash [x] : \square_r A \ \rhd\ \Delta}\ (\textsc{p}\square)$$

$- \neq r : R$ であり、仮定を満たさない。従って補題の結論は自明に成立する。

$$\text{Case.}\quad \frac{r \sqsubseteq 0 \quad D; \Sigma \vdash A : \mathsf{Type}}{D; \Sigma; r : R \vdash \_ : A \ \rhd\ \emptyset}\ (\textsc{p}\_)$$

最後のパターン型導出が (p_) の場合、$(\rhd\_)$ が適用できて

$$\frac{}{(t'' \rhd \_)t = t}\ (\rhd\_)$$

従って $t' = t$ とすれば補題の結論（進行性）を得る。今 $\Delta = \emptyset$ であり、補題の仮定に代入して $D; \Sigma; \Gamma_1 \vdash t : B$ を得る。これに (weak) を適用して

$$\frac{D; \Sigma; \Gamma_1 \vdash t : B}{D; \Sigma; \Gamma_1 + [\Gamma_2]_{0:R} \vdash t : B}\ (\rhd\_)$$

最後に、(p_) の前提から $r \sqsubseteq 0$ であることに注意すれば、補題の結論（保存性）を得る。 □

定理 A.4.5 [λ_{VL} の型安全性]
環境 D, Σ, Γ、項 t、型 A が存在して以下を満たす。

$$D; \Sigma; \Gamma \vdash t : A \Longrightarrow (\mathrm{value}\ t) \vee (\exists t', \Gamma'. t \rightsquigarrow t' \wedge D; \Sigma; \Gamma' \vdash t' : A \wedge \Gamma' \sqsubseteq \Gamma)$$

証明

$D; \Sigma; \Gamma \vdash t : A$ の型導出に関する構造的機能法で示す。最後に用いた型導出規則で場合分け。

$$\text{Case.}\quad \frac{\Sigma \vdash A : \mathsf{Type}}{D; \Sigma; x : A \vdash x : A}\ (\textsc{var})$$

変数 x は値である。従って定理の結論が成立する。

$$\text{Case.}\quad \frac{\begin{array}{c} D;\Sigma;-\vdash p : B_1 \rhd \Delta' \\ D;\Sigma;\Gamma, x : A', \Gamma', \Delta' \vdash t : B_2 \end{array}}{D;\Sigma;\Gamma, x : A', \Gamma' \vdash \lambda p.t : B_1 \to B_2}\ (\textsc{abs})$$

ラムダ抽象 $\lambda p.t$ は値である。従って定理の結論が成立する。

$$\text{Case.}\quad \frac{\begin{array}{c} D;\Sigma;\Gamma_1 \vdash t_1 : B \to A \\ D;\Sigma;\Gamma_2 \vdash t_2 : B \end{array}}{D;\Sigma;\Gamma_1 + \Gamma_2 \vdash t_1\ t_2 : A}\ (\textsc{app})$$

t_1 が値か否かで場合分け。

- t_1 が値の場合

  A.4.1 からある項 t_1' が存在して $t_1 = \lambda p.t_1'$ を満たす。これを用いて最後の導出は以下のように書き換えられる。

$$\frac{\dfrac{D;\Sigma;-\vdash p : B \rhd \Delta' \qquad D;\Sigma;\Gamma_1, \Delta' \vdash t_1' : A}{D;\Sigma;\Gamma_1 \vdash \lambda p.t_1' : B \to A}\ (\textsc{abs}) \qquad D;\Sigma;\Gamma_2 \vdash t_2 : B}{D;\Sigma;\Gamma_1 + \Gamma_2 \vdash (\lambda p.t_1')\ t_2 : A}\ (\textsc{app})$$

  このとき $(\textsc{p}\beta)$ が適用できて

$$\frac{}{\underbrace{(\lambda p.t_1')\ t_2}_{t} \rightsquigarrow (t_2 \rhd p)\ t_1'}\ (\textsc{p}\beta)$$

  ここで B が線形な型かコエフェクト付きの型で場合分け。

  - B が線形な型の場合

    A.4.3 から

$$\left.\begin{array}{r} D;\Sigma;\Gamma_2 \vdash t_2 : B \\ D;\Sigma;-\vdash p : B \rhd \Delta' \\ D;\Sigma;\Gamma_1, \Delta' \vdash t_1' : A \end{array}\right\} \implies \exists t_1''.\left\{\begin{array}{c} (t_2 \rhd p)\ t_1' = t_1'' \\ D;\Sigma;\Gamma_1 + \Gamma_2 \vdash t_1'' : A \end{array}\right.$$

    $t = t_1''$ とすれば定理の結論を得る。

  - B がコエフェクト付きの型の場合

    A.4.4 から

$$\left.\begin{array}{r} D;\Sigma;[\Gamma_2] \vdash t_2 : B \\ D;\Sigma;r : R \vdash p : B \rhd \Delta' \\ D;\Sigma;\Gamma_1, \Delta' \vdash t_1' : A \end{array}\right\} \implies \exists t_1''.\left\{\begin{array}{c} (t_2 \rhd p)\ t_1' = t_1'' \\ D;\Sigma;\Gamma_1 + r \cdot \Gamma_2 \vdash t_1'' : A \end{array}\right.$$

    $t = t_1''$ とすれば定理の結論を得る。

- t_1 が値でない場合

  $(\textsc{app})$ が適用でき、ある t_1' が存在して

$$\frac{t_1 \rightsquigarrow t_1'}{\underbrace{t_1\ t_2}_{t} \rightsquigarrow t_1'\ t_2}\ (\textsc{app})$$

を満たす。従って $t' = t_1'\ t_2$ とすればよい。（進行性）帰納法の仮定から最後の導出の前提に結論を定理の主張を適用できて $D; \Sigma; \Gamma_1 \vdash t_1' : B \to A$ であり、これに改めて (app) を適用すれば

$$\frac{D; \Sigma; \Gamma_1 \vdash t_1' : B \to A \quad D; \Sigma; \Gamma_2 \vdash t_2 : B}{D; \Sigma; \Gamma_1 + \Gamma_2 \vdash t_1'\ t_2 : A}\ (\textsc{app})$$

これにより定理の結論（保存性）を得る。

$$\text{Case.}\ \frac{}{D; \Delta; \emptyset \vdash n : \mathsf{Int}}\ (\textsc{int})$$

整数値 n は値である。従って定理の結論は自明に成立する。

$$\frac{(x : \forall\{\overrightarrow{\alpha : \kappa}\}.A) \in D \quad \theta, \Sigma' = \mathsf{inst}(\overrightarrow{\alpha : \kappa})}{D; \Sigma, \Sigma'; \emptyset \vdash x : \theta A}\ (\mathrm{C})$$

トップレベル定義 x は値である。従って定理の結論は自明に成立する。

$$\text{Case.}\ \frac{\begin{array}{c}\Sigma \vdash R : \mathsf{Coeff} \\ D; \Sigma; \Gamma_1 \vdash t : A\end{array}}{D; \Sigma; \Gamma_1 + [\Delta']_{0:R} \vdash t : A}\ (\textsc{weak})$$

最後の導出が (weak) のとき、導出の前後で項 t は不変である。t が値の場合は直ちに定理の主張を満たすので、t が値でない場合をかんがえればよい。帰納法の仮定からある t' が存在して

$$t \rightsquigarrow t' \wedge D; \Sigma; \Gamma_1 \vdash t' : A$$

を満たす。（進行性）これに改めて (weak) を適用して

$$\frac{\begin{array}{c}\Sigma \vdash R : \mathsf{Coeff} \\ D; \Sigma; \Gamma_1 \vdash t' : A\end{array}}{D; \Sigma; \Gamma_1 + [\Delta']_{0:R} \vdash t' : A}\ (\textsc{weak})$$

以上より定理の結論（保存性）を得る。

$$\text{Case.}\ \frac{\begin{array}{c}\Sigma \vdash R : \mathsf{Coeff} \\ D; \Sigma; \Gamma_1, x : B \vdash t : A\end{array}}{D; \Sigma; \Gamma_1, x : [B]_{1:R} \vdash t : A}\ (\textsc{der})$$

最後の導出が (der) のとき、導出の前後で項 t は不変である。t が値の場合は直ちに定理の主張を満たすので、t

が値でない場合をかんがえればよい。帰納法の仮定からある t' が存在して

$$t \rightsquigarrow t' \wedge D; \Sigma; \Gamma_1, x : B \vdash t' : A$$

を満たす。（進行性）これに改めて (DER) を適用して

$$\frac{\Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; \Gamma_1, x : B \vdash t' : A}{D; \Sigma; \Gamma_1, x : [B]_{1:R} \vdash t' : A}\ (\textsc{der})$$

以上より定理の結論（保存性）を得る。

$$\text{Case.}\quad \frac{\Sigma \vdash r : \uparrow R \quad \Sigma \vdash R : \mathrm{Coeff} \qquad D; \Sigma; [\Gamma] \vdash t : B}{D; \Sigma; r \cdot [\Gamma] \vdash [t] : \square_r B}\ (\textsc{pr})$$

box 項 $[t]$ は値である。従って定理の結論は自明に成立する。

$$\text{Case.}\quad \frac{D; \Sigma; [\Gamma_i] \vdash t_i : A}{D; \Sigma; \bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \{\overline{l = t} | l_i\} : \square_{\overline{l}} A}\ (\textsc{ver})$$

Versioned Value $\{\overline{l = t} | l_i\}$ は値である。従って定理の結論は自明に成立する。

$$\text{Case.}\quad \frac{D; \Sigma; [\Gamma_i] \vdash t_i : A}{D; \Sigma; \bigcup_i (\{l_i\} \cdot [\Gamma_i]) \vdash \langle \overline{l = t} | l_i \rangle : A}\ (\textsc{veri})$$

最後の導出が (VERI) のとき、(VER) が適用できて

$$\frac{}{\langle \overline{l = t} | l_i \rangle \rightsquigarrow t_i @ l_i}\ (\textsc{ver})$$

A.4.2 から、ある t' が存在して

$$\left.\begin{matrix} D; \Sigma; [\Gamma_i] \vdash t_i : A \\ l_i \in r \end{matrix}\right\} \implies \exists t'. \begin{cases} t_i @ l_i = t' \\ D; \Sigma; \{l_i\} \cdot [\Gamma_i] \vdash t' : A \end{cases}$$

以上より定理の主張が成立する。

$$\text{Case.}\quad \frac{D; \Sigma; \Gamma \vdash t_1 : \square_r A \quad l \in r}{D; \Sigma; \Gamma \vdash t_1.l : A}\ (\textsc{rstr})$$

最後の導出が (RSTR) の場合、t_1 が値か否かで場合分け。

- t_1 が値の場合

  A.4.1 より t_1 には $[t'']$ か Versioned Value のいずれか一方である。それぞれで場合分け。

- $t_1 = [t'']$ の場合

  最後の型導出は以下のように書き換えられる。

$$\dfrac{\dfrac{D;\Sigma;[\Gamma'] \vdash t'' : A \qquad \Sigma \vdash r : R \qquad \Sigma \vdash R : \mathrm{Coeff}}{D;\Sigma;r \cdot [\Gamma'] \vdash [t''] : \square_r A}\,(\textsc{pr}) \qquad l \in r}{D;\Sigma;r \cdot [\Gamma'] \vdash [t''].l : A}\,(\textsc{rstr})$$

  このとき (EXTR1) が適用でき、

$$\dfrac{}{[t''].l \rightsquigarrow t''@l}\,(\textsc{extr1})$$

  A.4.2 から、ある t' が存在して

$$\left.\begin{array}{r} D;\Sigma;[\Gamma'] \vdash t'' : A \\ l \in r \end{array}\right\} \implies \exists t'. \left\{\begin{array}{c} t''@l = t' \\ D;\Sigma;\{l\} \cdot [\Gamma'] \vdash t' : A \end{array}\right.$$

  以上より定理の主張が成立する。

- $t_1 = \{\overline{l = t} | l'\}$ の場合

  最後の型導出は以下のように書き換えられる。

$$\dfrac{\dfrac{D_i;\Sigma_i;[\Gamma_i] \vdash t_i : A \qquad \Sigma \vdash r : R \qquad \Sigma \vdash R : \mathrm{Coeff}}{D;\Sigma;\Sigma_i\{l_i\} \cdot [\Gamma_i] \vdash \{\overline{l = t} \mid l'\} : \square_r A}\,(\textsc{pr}) \qquad l \in r}{D;\Sigma;\Sigma_i\{l_i\} \cdot [\Gamma_i] \vdash \{\overline{l = t} \mid l'\}.l : A}\,(\textsc{rstr})$$

  このとき (EXTR2) が適用でき、

$$\dfrac{}{\{\overline{l = t} \mid l'\}.l \rightsquigarrow t_l@l}\,(\textsc{extr2})$$

  A.4.2 から、ある t' が存在して

$$\left.\begin{array}{r} D;\Sigma;[\Gamma'] \vdash t_l : A \\ l \in r \end{array}\right\} \implies \exists t'. \left\{\begin{array}{c} t_l@l = t' \\ D;\Sigma;\{l\} \cdot [\Gamma'] \vdash t' : A \end{array}\right.$$

  以上より定理の主張が成立する。

- t_1 が値でない場合

  (RSTR) が適用でき、以下を得る。

$$\dfrac{t_1 \rightsquigarrow t_1'}{t_1.l \rightsquigarrow t_1'.l}\,(\textsc{rstr})$$

  $t' = t_1'.l$ とすれば定理の結論（進行性）が成立する。また帰納法の結論から定理を前提に適用し、改めて (RSTR) を適用することで以下を得る。

$$\dfrac{D;\Sigma;\Gamma \vdash t_1' : \square_r A \quad l \in r}{D;\Sigma;\Gamma \vdash t_1'.l : A}\,(\textsc{rstr})$$

  以上より定理の結論（保存性）を得る。

Case. $\dfrac{D;\Sigma;\Gamma_1, x : [B]_r, \Gamma_2 \vdash t : A \quad r \sqsubseteq s}{D;\Sigma;\Gamma_1, x : [B]_s, \Gamma_2 \vdash t : A}\,(\sqsubseteq)$

最後の導出が (⊑) の場合、導出の前後で項 t は不変である。t が値の場合は直ちに定理の主張を満たすので、t が値でない場合を考えればよい。帰納法の仮定からある t' が存在して

$$t \rightsquigarrow t' \wedge D; \Sigma; \Gamma_1, x : [B]_r, \Gamma_2 \vdash t' : A$$

を満たす。（進行性）これに改めて (⊑) を適用して

$$\frac{D; \Sigma; \Gamma_1, x : [B]_r, \Gamma_2 \vdash t' : A \quad r \sqsubseteq s}{D; \Sigma; \Gamma_1, x : [B]_s, \Gamma_2 \vdash t' : A} \ (\sqsubseteq)$$

以上より定理の結論（保存性）を得る。 □