コンパイルとコード検証の証明論的アプローチ

大堀 淳

JAIST

ohori@jaist.ac.jp
http://www.jaist.ac.jp/~ohori

研究が目指すもの

高信頼プログラミング言語の基礎の確立

高信頼プログラミング言語が基礎としうるもの：

「型理論」

- 堅牢な言語設計,
- プログラムの整合性の検証,

より一般的には,

「数理論理学の証明論」

- 堅牢な言語設計,
- プログラムの整合性の検証,
- プログラムの操作的意味論
- 系統的なコンパイル

計算機科学の基礎としての「論理学」

「論理学」の一般的な理解

思考の形式に関する学

より技術的定義

言語使用に関する学

「言語」の定義

有限のシンボルと有限の規則からなる形式体系

数理論理学の証明論

- 言語の構造（ものの名前，構文）
- 文（証明）の構成規則（構文構成規則，名前の参照関係）
- 文の意味（同値関係，表示的意味）
- 証明体系間の同一性（ヒルベルトシステム，自然演繹，LK）

これは以下を含むプログラミング言語の直接の基礎となりうる．

- 堅牢な言語設計，
- プログラムの整合性の検証，
- プログラムの操作的意味論
- 系統的なコンパイル

研究の動機と目的

プログラミング言語実装の基礎である

- 低レベルコード
- コンパイル過程

の各項目に対する証明論的解釈を構築することにより,

プログラミング言語の証明論的基礎を確立する

これらを通じ,

1. 高水準言語の堅牢で正しいコンパイラの構築
2. 低レベルコードの検証や分析システムの構築

への応用を目指す.

1. 堅牢で正しいコンパイラの構築

ソフトウエアの正しさは，コンパイラの正しさに直接依存

しかし，コンパイルとコードに関する理論が未成熟のため コンパイラの構築はad-hocに行われているのが現状.

論理学的アプローチでは,

- 機械語コードは証明システムとして，また,
- コンパイルは証明変換として表現される.

これによって，コンパイラは，再帰的は証明変換として定義でき，種々の望ましい性質を検証可能.

2. コードの検証と分析

実行可能コードが動的にロード・リンク・実行される

インターネットプログラミング環境において，コードの検証と分析はシステムの信頼性と安全性にとって枢要の技術

本研究が提唱する論理学的アプローチは,

- コードからの証明の再構築,
- 証明変換により高水準言語への逆変換

等を可能にし，Proof Carrying Code などとはを補完／代替する，コードの検証と分析にとっての新たなアプローチとなりうる.

本アプローチの背景 (I)

高水準プログラミング言語のモデルとしての型付きラムダ計算

項の集合:

$$M ::= c^b \mid x \mid \lambda x:\tau.M \mid M\ M \mid (M,M) \mid M.1 \mid M.2 \mid inl(M) \mid inr(M) \mid \mathbf{case}\ M\ \mathbf{of}\ x.M, x.M$$

例：

```
function f(x,y) = x + y
...
```

は以下のような項として表現される.

$$(\lambda f.\cdots)\ (\lambda x.x.1 + x.2)$$

型付きラムダ計算

型の集合:

$$\tau ::= b \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau$$

型付け規則:

- データ構造の生成

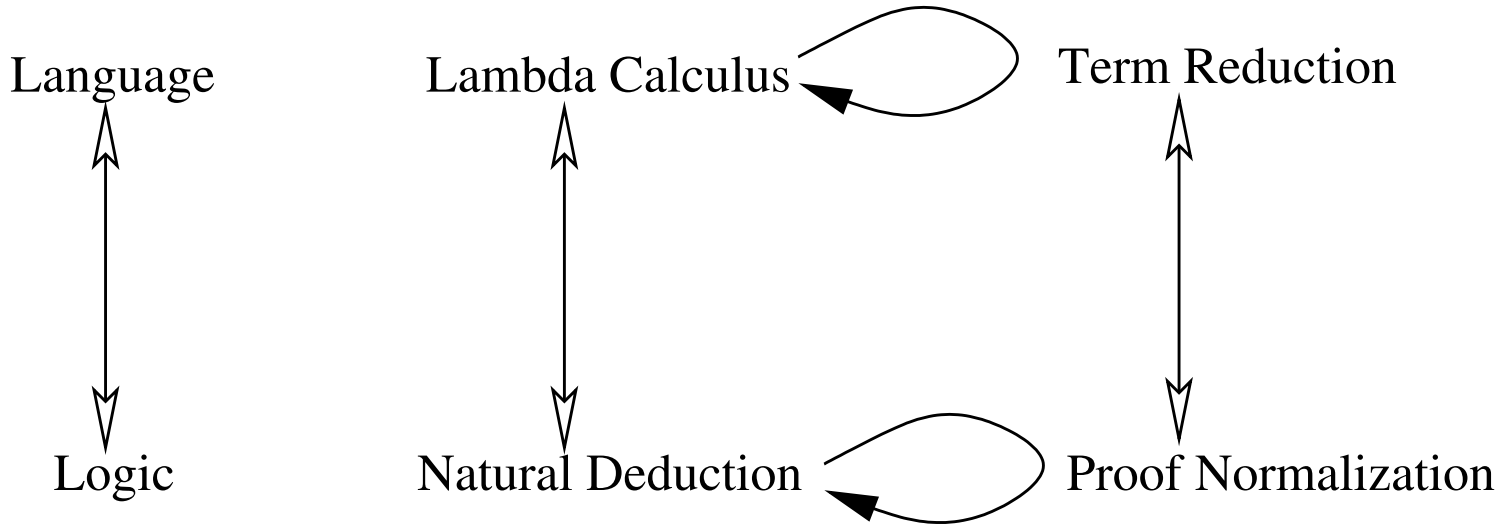
$$\frac{\Gamma \vdash M_1 : \tau_1 \quad \Gamma \vdash M_2 : \tau_2}{\Gamma \vdash (M_1, M_2) : \tau_1 \times \tau_2} \qquad \frac{\Gamma; x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . M : \tau_1 \to \tau_2}$$

- データ構造の利用

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash M.1 : \tau_1} \qquad \frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1\ M_2 : \tau_2}$$

本アプローチの背景 (II)

ラムダ計算に対する Curry-Howard 同型関係



- プログラム：型 $\Longleftrightarrow$ $\dfrac{\text{証明}}{\text{命題}}$
- 項の簡約 $\Longleftrightarrow$ 証明の正規化

自然演繹システム

命題集合:

$$A ::= p^b \mid A \supset A \mid A \wedge A \mid A \vee A$$

証明規則:

- 導出規則

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \qquad \frac{\Gamma; A \vdash B}{\Gamma \vdash A \supset B}$$

- 除去規則

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B}$$

型 $\Longleftrightarrow$ 命題

$$
\begin{aligned}
b &\Longleftrightarrow p^b \\
\tau \to \sigma &\Longleftrightarrow A \supset B \\
\tau \times \sigma &\Longleftrightarrow A \wedge B \\
\tau + \sigma &\Longleftrightarrow A \vee B
\end{aligned}
$$

プログラム：型 $\Longleftrightarrow$ 証明：命題

$$
\begin{aligned}
\Gamma \vdash c^b : b &\Longleftrightarrow \Gamma \vdash p^b \\
\Gamma; x : \tau \vdash x : \tau &\Longleftrightarrow \Gamma; A \vdash A \\
\frac{\Gamma; x : \tau \vdash M : \tau}{\Gamma \vdash \lambda x : \tau . M : \tau \to \sigma} &\Longleftrightarrow \frac{\Gamma; A \vdash B}{\Gamma \vdash A \supset B} \\
&\vdots
\end{aligned}
$$

項の簡約 $\Longleftrightarrow$ 証明正規化

$$\left(\begin{array}{c} \dfrac{\dfrac{\begin{array}{c}[\Gamma; x:\tau \vdash x : \tau] \\ \vdots \\ \Gamma; x:\tau \vdash M_1 : \sigma\end{array}}{\Gamma \vdash \lambda x:\tau.M_1 : \tau \to \sigma} \quad \begin{array}{c}\vdots \\ \Gamma \vdash M_2 : \tau\end{array}}{\Gamma \vdash (\lambda x:\tau.M_1)\ M_2 : \sigma} \\ \longrightarrow \quad \begin{array}{c}\vdots \\ \Gamma \vdash M_2 : \tau \\ \vdots \\ \Gamma \vdash [M_2/x]M_1 : \sigma\end{array} \end{array}\right)$$

$$\Longleftrightarrow \quad \left(\begin{array}{c} \dfrac{\dfrac{\begin{array}{c}[\Gamma; A \vdash A] \\ \vdots \\ \Gamma; A \vdash B\end{array}}{\Gamma \vdash A \supset B} \quad \begin{array}{c}\vdots \\ \Gamma \vdash A\end{array}}{\Gamma \vdash B} \\ \longrightarrow \quad \begin{array}{c}\vdots \\ \Gamma \vdash A \\ \vdots \\ \Gamma \vdash B\end{array} \end{array}\right)$$

この洞察は,

- より精密で洗練された型システム
- 継続計算などの論理学的分析と理解

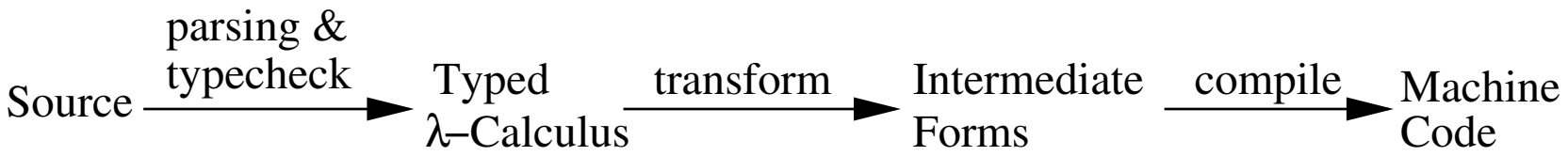
  ⋮

など，プログラミング言語の設計等に重要な成果をもたらした.
しかし,

この対応関係はプログラミング言語の実態を必ずしも反映していない！

プログラミング言語処理系の構造

1. 構文論的分析 ＋ 型理論的分析
2. 中間言語をへの変換，および機械語コード生成，
3. メモリーリソース（レジスタ，スタック，ヒープ）を用いたコードの実行



たとえば,

$$(\lambda x.\lambda y.(x,y))\ 1\ 2$$

のような式は,

```
let $3 = (fn x=> (fn y=> (x,y) ))
in app ($3 1) is $2
   in app ($2 2) is $1
      in $1 end
   end
end
```

以下のような中間言語（A-Normal式）に変換され，さらに,

```
     r0 <- Code(L1)
     r1 <- Const(1)
     r2 <- Const(2)
     r0 <- call r0(r1,r2)
     Return(r0)
L1:  r2 <- Pair(r1,r0)
     Return(r2)
```

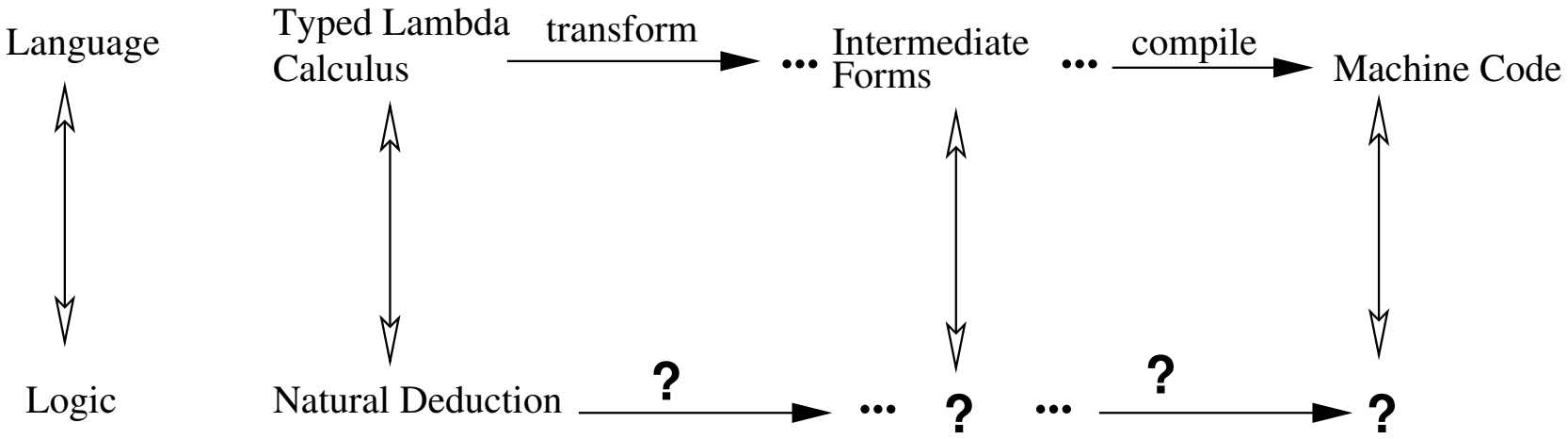
や

```
     Code(L1)
     Const(1)
     Const(2)
     Call(2)
     Return
L1:  Acc(0)
     Acc(1)
     Pair
     Return
```

ようなのなどの形式の低レベルコードに変換される.

コンパイルとコード分析のための論理学的アプローチ

コンパイル: 以下のダイヤグラムを完成させる

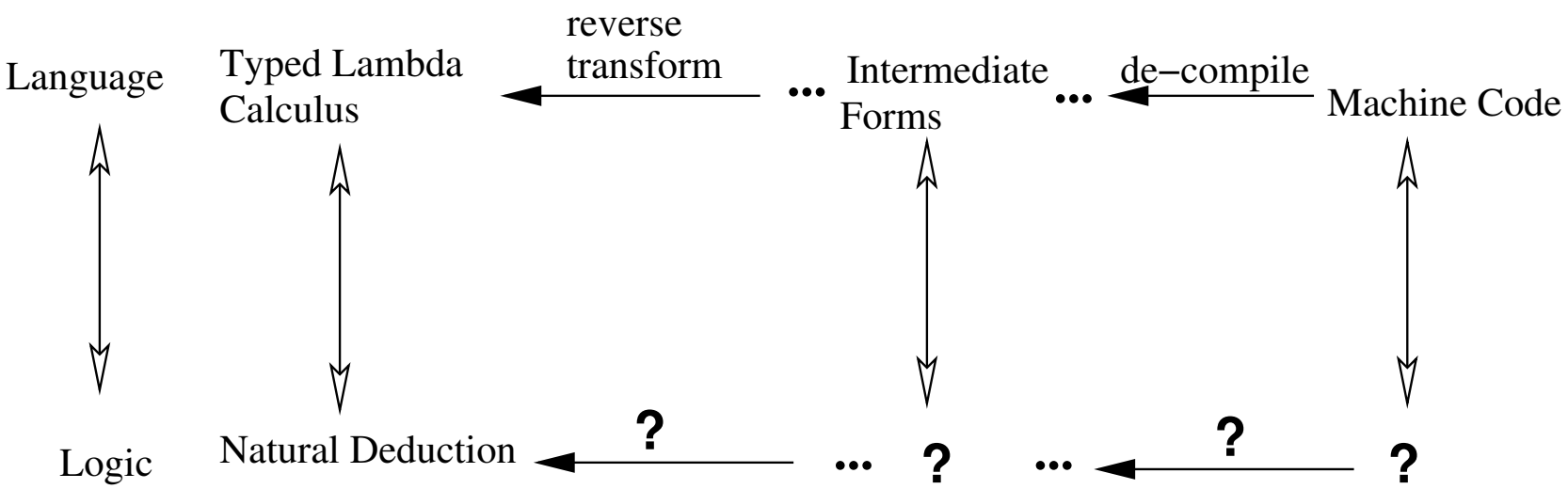


これにより,

- 種々の中間言語および機械語コード
- コンパイル過程

に関する論理学的基礎が与えられる.

コードの分析と検証: 逆のダイヤグラムを完成させる



これにより,

- コード分析
- 逆コンパイル
- コード変換

などの基礎が与えられる.

このアプローチの基礎となるアイデア

- 各中間言語：より詳細な制御構造からなる証明システム
- 機械語命令：機械による解釈に向く証明規則,
- コンパイル／逆コンパイル：これら証明システム間の証明変換.

洞察: 自然演繹は必ずしも機械向けではない.

- 除去規則は，任意で不特定の形の証明に対して演算を適用：

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash M.1 : \tau_1}$$

- 導出規則は，任意に大きな項を結合：

$$\frac{\Gamma \vdash M_1 : \tau_1 \times \tau_1 \qquad \Gamma \vdash M_2 : \tau_1 \times \tau_2}{\Gamma \vdash (M_1, M_2) : \tau_1 \times \tau_2}$$

コンパイル過程

- A-Normal式：除去規則を持たない証明体系.
  - A-normal形式：ある種のシーケント計算系
  - translation：除去規則 $\Longrightarrow$ 左規則と + カット規則に変換
- バイトコード / 疑似コード：右規則のない証明体系
  - 疑似命令：一つの前提をもつ左規則
  - コード生成：右規則の知りシリアライズ.
- 機械語コード：暗黙の構造規則を持たない証明体系
  - レジスタ操作命令：明示的構造規則
  - レジスタ割り付け：構造規則導入

逆コンパイル

上記のそれぞれの逆変換の合成.

コンパイルに関する文献

- (疑似コード)
  Ohori, A. The Logical Abstract Machine: a Curry-Howard isomorphism for machine code. FLOPS99, LNCS 1722, 1999.
- (A-normal 形式)
  A. Ohori. A Curry-Howard Isomorphism for Compilation and Program Execution. In Proc. Typed Lambda Calculi and Applications, LNCS 1581, 1999.
- (機械語コード，レジスタ割り当て)
  A. Ohori. Register Allocation by Proof Transformation. In Proc. ESOP, 2003. (to appear in J. Scinece of Computer Softoware)
- (パターンマッチングコンパイル（投稿中)
  A. Ohori and S. Osaka. A Fresh Look at Pattern Matching Compilation

コード分析

- (JAVAバイトコードの証明システム)
  T. Higuchi and A. Ohori. Java bytecode as typed term calculus. In Proc. ACM PPDP Conference, 2002
- (JAVAバイトコードの逆コンパイル)
  S. Katsumata and A. Ohori, Proof-Directed De-Compilation of Low-Level Code. In Proc. ESOP Symposiun, LNCS, 2001.
- (JAVAバイトコードのアクセス制御方式)
  T. Higuchi and A. Ohori. A Static Type System for JVM Access Control. Proc. ACM ICFP Conference, pages 227 - 237, 2003.

A-Normal形式／A-Normal変換の証明論

A-Normal式

$$
\begin{aligned}
V &::= c^b \mid x \mid \lambda x.M \mid (V,V) \mid inl(V) \mid inr(V) \\
M &::= V \mid \mathbf{app}\ (x\ V)\ \mathbf{is}\ y\ \mathbf{in}\ M \mid \\
&\quad\ \mathbf{proj}\ x\ \mathbf{on}\ (y,z)\ \mathbf{in}\ M \mid \\
&\quad\ \mathbf{case}\ x\ \mathbf{of}\ y.M, z.M \mid \mathbf{let}\ x = V\ \mathbf{in}\ M
\end{aligned}
$$

A-Normal式の特徴

- 操作は変数に対してのみ
- 全ての計算結果には名前が付く

$\Longrightarrow$ シーケント計算系の規則の特徴と一致.

Kleeneのシーケント計算系： G3

$$\frac{\Delta; A \vdash B}{\Delta \vdash A \supset B} \qquad \frac{\Delta; A \supset B \vdash A \quad \Delta; A \supset B; B \vdash C}{\Delta; A \supset B \vdash C}$$

$$\frac{\Delta \vdash A \quad \Delta \vdash B}{\Delta \vdash A \wedge B} \qquad \frac{\Delta; A \wedge B; A; B \vdash C}{\Delta; A \wedge B \vdash C}$$

$$\frac{\Delta \vdash A_i}{\Delta \vdash A_1 \vee A_2} \qquad \frac{\Delta; A \vee B; A \vdash C \quad \Delta; A \vee B; B \vdash C}{\Delta; A \vee B \vdash C}$$

$$\frac{\Delta \vdash A \quad \Delta; A \vdash B}{\Delta \vdash B}$$

左規則： 変数への操作の適用

A-normal式の型付け規則

$$\frac{\Gamma; x : \tau_1 \to \tau_2 \vdash M_1 \ : \ \tau_1 \quad \Gamma; x : \tau_1 \to \tau_2;\ y : \tau_2 \vdash M_2 \ : \ \tau_3}{\Gamma; x : \tau_1 \to \tau_2 \vdash \mathbf{app}\ (x\ M_1)\ \mathbf{is}\ y\ \mathbf{in}\ M_2 \ : \ \tau_3}$$

$$\frac{\Gamma; x : \tau_1 \times \tau_2;\ y : \tau_1;\ z : \tau_2 \vdash M \ : \ \tau_3}{\Gamma; x : \tau_1 \times \tau_2 \vdash \mathbf{proj}\ x\ \mathbf{on}\ (y, z)\ \mathbf{in}\ M \ : \ \tau_3}$$

$$\frac{\Gamma; x : \tau_1 + \tau_2;\ y : \tau_1 \vdash M_1 \ : \ \tau_3 \quad \Gamma; x : \tau_1 + \tau_2;\ z : \tau_2 \vdash M_2 \ : \ \tau_3}{\Gamma; x : \tau_1 + \tau_2 \vdash \mathbf{case}\ x\ \mathbf{of}\ y.M_1, z.M_2 \ : \ \tau_3}$$

$$\frac{\Gamma \vdash M_1 \ : \ \tau_1 \quad \Gamma; x : \tau_1 \vdash M_2 \ : \ \tau_2}{\Gamma \vdash \mathbf{let}\ x = M_1\ \mathbf{in}\ M_2 \ : \ \tau_2}$$

A-normal変換:

除去規則を左規則とカット規則に変換.

例：∧除去規則

$$\frac{\Delta \vdash A \wedge B}{\Delta \vdash A}$$

は以下のように変換する

$$\frac{\Delta \vdash A \wedge B \qquad \dfrac{\dfrac{}{\Delta; A \wedge B; A; B \vdash A}}{\Delta; A \wedge B \vdash A}}{\Delta \vdash A} \quad .$$

これは，組の要素の取出し操作を

$$\frac{\Gamma \vdash M \ : \ \tau_1 \times \tau_2}{\Gamma \vdash M.1 \ : \ \tau_1}$$

以下のように変換することに相当

$$\frac{\Gamma \vdash M' \ : \ \tau_1 \times \tau_2 \quad \dfrac{\dfrac{}{\Gamma; y : \tau_1 \times \tau_2;\ x_1 : \tau_1;\ x_2 : \tau_2 \vdash x_1 \ : \ \tau_1}}{\Gamma; y : \tau_1 \times \tau_2 \vdash \mathbf{proj}\ y\ \mathbf{on}\ (x_1, x_2)\ \mathbf{in}\ x_1 \ : \ \tau_1}}{\Gamma \vdash \mathbf{let}\ y = M'\ \mathbf{in}\ \mathbf{proj}\ y\ \mathbf{on}\ (x_1, x_2)\ \mathbf{in}\ x_1 \ : \ \tau_1}$$

疑似コードの論理学

疑似コードの特徴

- 命令の逐次実行
  $\Longrightarrow$ 分岐のない証明規則
- 各命令はレジスタやスタックの変更操作
  $\Longrightarrow$ 一つの上式（前提）からなる左規則
- 汎用なメモリ操作命令
  $\Longrightarrow$ 各規則は結論の論理式を変更しない.
- コンパイルされたコードを呼び出し機構を装備
  $\Longrightarrow$ 既存の証明を参照するメタレベル機構

コードの論理学

- 各命令 $I : \Delta \Longrightarrow \Delta'$ は以下の形の左規則に対応

$$\frac{\Delta' \vdash C}{\Delta \vdash C} \quad \text{(rule-}I\text{), and}$$

- return 命令 は，証明系の始式にに対応

$$\Delta; C \vdash C \quad \text{(return)}$$

- さらに，コンパイル済のコードを呼び出す以下のメタレベル規則を持つ:

$$\frac{\Delta; (\Delta_0 \Rightarrow A) \vdash C}{\Delta \vdash C} \quad \text{if } \Delta_0 \vdash A$$

命令の例:

$$\frac{\Delta; A \vdash C}{\Delta; A \wedge B \vdash C}\ (\mathsf{Fst}) \qquad \frac{\Delta; A \wedge B \vdash C}{\Delta; A; B \vdash C}\ (\mathsf{Pair})$$

各規則が $I_i : \Delta_i \Longrightarrow \Delta_{i+1}$ である

$$I_1; \cdots; I_n; \mathit{Return}$$

のようなプログラムは，以下のような証明に対応:

$$\begin{array}{l} \dfrac{\dfrac{}{\Delta_{n+1}; C \vdash C}\ (\mathsf{return})}{\Delta_n \vdash C}\ (\mathsf{rule}\text{-}I_n) \\ \qquad\vdots \\ \dfrac{\Delta_2 \vdash C}{\Delta_1 \vdash C}\ (\mathsf{rule}\text{-}I_1) \end{array}$$

コードの実行は，最後の式を除去に対応.

Sequential Sequent Calculus

$\Delta; A \vdash A$

$$\frac{\Delta; B \vdash A}{\Delta \vdash A}\ (B \in \Delta) \qquad \frac{\Delta; B \vdash A}{\Delta \vdash A}\ (B \text{ an axiom})$$

$$\frac{\Delta; B \wedge C \vdash A}{\Delta; B; C \vdash A} \qquad \frac{\Delta; B \vdash A}{\Delta; B \wedge C \vdash A}$$

$$\frac{\Delta; L \vdash A}{\Delta; B \wedge C \vdash A} \qquad \frac{\Delta; B \vee C \vdash A}{\Delta; B \vdash A}$$

$$\frac{\Delta; B \vee C \vdash A}{\Delta; C \vdash A} \qquad \frac{\Delta; C \vdash A}{\Delta; B_1 \vee B_2 \vdash A}\ (\text{if } \Delta; B_i \vdash C,\ i = 1, 2)$$

$$\frac{\Delta; (\Delta_0 \Rightarrow A_0) \vdash A}{\Delta \vdash A} \quad (\text{if } \Delta_0 \vdash A_0)$$

$$\frac{\Delta; A_0 \vdash A}{\Delta; (\Delta_0 \Rightarrow A_0); \Delta_0 \vdash A}$$

$$\frac{\Delta; (\Delta_1 \Rightarrow A_0) \vdash A}{\Delta; (\Delta_0; \Delta_1 \Rightarrow A_0); \Delta_0 \vdash A}$$

Δ を機械メモリーの状態と見なせば，この証明システムはコード言語の型システムに対応する．

- スタック機械： Δ 命題リスト
- レジスタ機械： Δ 名前付き命題集合

スタックベースコード

$$I ::= \mathsf{Return} \mid \mathsf{Acc}(n) \mid \mathsf{Const}(c^b) \mid \mathsf{Pair} \mid \mathsf{Fst} \mid \mathsf{Snd} \mid \mathsf{Inl} \mid \mathsf{Inr} \mid \mathsf{Case} \mid \mathsf{Code}(L) \mid \mathsf{Call(n)} \mid \mathsf{App}(n)$$

$$\Delta; \tau \vdash \mathsf{Return}(n) : \tau$$

$$\frac{\Delta; \tau_1 \vdash L : \tau}{\Delta \vdash \mathsf{Acc}(n); L : \tau} \ (\tau_1 \in \Delta) \qquad \frac{\Delta; b \vdash L : \tau}{\Delta \vdash c^b; L : \tau}$$

$$\frac{\Delta; \tau_1 \wedge \tau_2 \vdash L : \tau}{\Delta; \tau_1; \tau_2 \vdash \mathsf{Pair}; L : \tau} \qquad \frac{\Delta; \tau_1 \vdash L : \tau}{\Delta; \tau_1 \wedge \tau_2 \vdash \mathsf{Fst}; L : \tau}$$

$$\frac{\Delta; \tau_2 \vdash L : \tau}{\Delta; \tau_1 \wedge \tau_2 \vdash \mathsf{Snd}; L : \tau} \qquad \frac{\Delta; \tau_1 \vee \tau_2 \vdash L : \tau}{\Delta; \tau_1 \vdash \mathsf{Inl}; L : \tau}$$

$$\frac{\Delta;\tau_1 \vee \tau_2 \vdash L \ : \ \tau}{\Delta;\tau_2 \vdash \mathsf{Inr};L \ : \ \tau}$$

$$\frac{\Delta;\tau_3 \vdash L \ : \ \tau}{\Delta;\tau_1 \vee \tau_2 \vdash \mathsf{Case}(L_1,L_2);\mathsf{L} : \ \tau} \ (\text{if } \Delta;\tau_i \vdash L_1 \ : \ \tau_3,\ i \in \{1,2\})$$

$$\frac{\Delta;(\Delta_0 \Rightarrow \tau_0) \vdash L \ : \ \tau}{\Delta \vdash \mathsf{Code}(L');L \ : \ \tau} \quad (\text{if } \Delta_0 \vdash L' \ : \ \tau_0)$$

$$\frac{\Delta;\tau_0 \vdash L \ : \ \tau}{\Delta;(\Delta_0 \Rightarrow \tau_0);\Delta_0 \vdash \mathsf{Call}(n);L \ : \ \tau}$$

$$\frac{\Delta;(\Delta_1 \Rightarrow \tau_0) \vdash L \ : \ \tau}{\Delta;(\Delta_0;\Delta_1 \Rightarrow \tau_0);\Delta_0 \vdash \mathsf{App}(n);L \ : \ \tau}$$

レジスタ割り付けの論理学

レジスタ割り付け問題: 疑似コード

$$
\begin{aligned}
x &::= \text{unbounded number of variables} \\
I &::= x = x \mid x = c \mid x = x\ op\ x \mid \text{if } x \text{ goto } l \\
B &::= \text{return}(x) \mid \text{goto } l \mid I; B \\
P &::= \{l : B, \ldots, l : B\}
\end{aligned}
$$

を，その意味いを変えずに，以下のコード言語に変換せよ.

$$
\begin{aligned}
r &::= \text{fixed number of registers} \\
x &::= \text{unbounded memory locations} \\
I &::= r = r \mid r = c \mid r = r \text{ op } r \mid \text{if } r \text{ goto } l \\
&\quad \mid \text{load } r, x \mid \text{store } r, x \\
B &::= \text{return}(x) \mid \text{goto } l \mid I; B \\
P &::= \{l : B, \ldots, l : B\}
\end{aligned}
$$

論理学的解釈の基本的なアイデア

レジスタ操作命令を構造規則と解釈.

たとえば,

$$\text{(Weaken-L)}\quad \frac{\Gamma \vdash \tau_0}{\Gamma, \tau \vdash \tau_0} \iff \frac{\Gamma \vdash B : \tau_0}{\Gamma, x : \tau \vdash \mathsf{discard}\ x; B : \tau_0}$$

$$\text{(ContractTrue-L)}\quad \frac{\Gamma, \tau \vdash \tau_0}{\Gamma \vdash \tau_0} \iff \frac{\Gamma, x : \tau \vdash B : \tau_0}{\Gamma \vdash \mathsf{alloc}\ x; B : \tau_0}$$

その下で，レジスタ割り当てを

> 構造規則を暗黙に含む証明システムから，構造規則を明示的に含む証明システムへの証明変換

と解釈する.

技術的展開

1. レジスタの生存期間を型として持つ証明システムを構築
2. 命令コードから証明を再構築（型推論）
3. 証明の正規化と最適化により，weakening rules と contraction rules を明示的に含む証明系に変換.
4. 証明系を，２つの環境を持つ証明系に変換

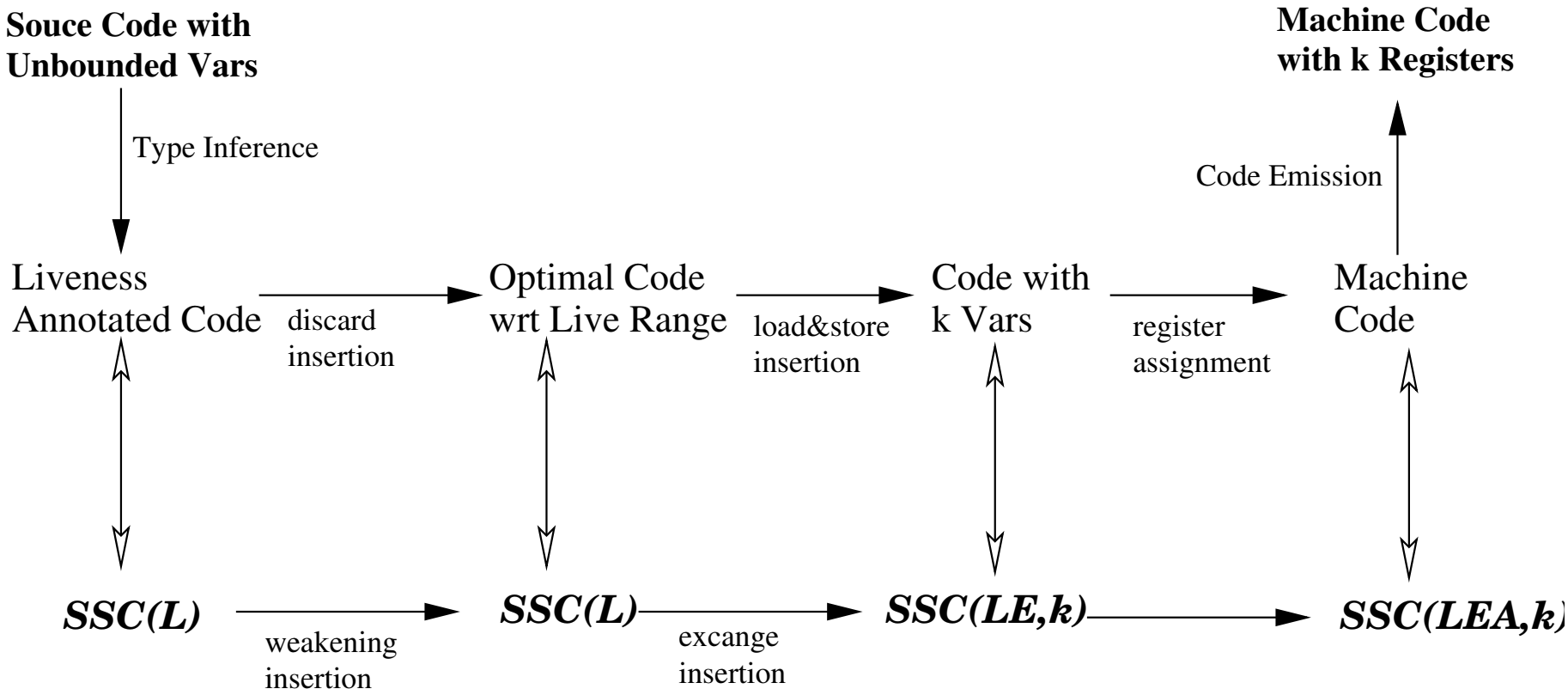
$$\Gamma \vdash B : \tau \quad \Longrightarrow \quad \Delta \mid \Pi \vdash_k B : \tau$$

ここで，Δ はメモリー環境，Π は **レジスタ環境**を表す.
exchange rules を明示的に含む証明系への変換に対応

5. さらに各生存期間型に番号を割り当て，コードを生成する.

$$\frac{\Delta \mid \Pi, x : t[p] \vdash_k B : \tau}{\Delta \mid \Pi, x : \mathsf{nil} \vdash_k \mathsf{alloc}\ (p, x); B : \tau}$$

レジスタ割り付けのCurry-Howard同型関係



さらなる研究課題

- ヒープ等の資源の表現
- 種々の言語機能の表現
  多相性，再帰 etc
- 他の体系との比較
  Proof-Carrying Code, TAL, バイトコード検証システム等
- コンパイルの最適化
  クロージャ変換，複数引数の最適化，等
- 他の技術への応用
  コード変換，Just-In-Time Compilation 等

THANK YOU FOR INVITING ME.

I seek collaboration on all aspects on programming languages, especially

- ML系言語のコンパイラの実装，プログラミング環境
- 高信頼開発環境
- 型理論，論理学的基礎

⋮

If you are interested, please contact at:

ohori@jaist.ac.jo
http://www.jaist.ac.jp/ ohori/