日本ソフトウェア科学会第 26 回大会 (2009 年度) 講演論文集

構造的多相性をもった言語の検証つきインタープリタ

Jacques Garrigue

Objective Caml の型システムは他言語と異なる機能を多くもっている．オブジェクト型や多相ヴァリアント型は構造的多相型によって実現されているが，その複雑さが実装の正しさの保証を難しくする．本研究では再帰型を含んだ構造的多相性を追加した Core ML に対して完全に証明されたインタープリタを Coq で実装した．Core ML と比べて証明の基本方針を変える必要はなかったが，全体的にかなり複雑になっている．評価の安全性以外に，型推論の健全性と完全性も証明し，全てのアルゴリズムがプログラムとして抽出でき，実際に実行できる．

1 はじめに

定理証明支援系によるプログラミング言語の基礎理論の定式化が進歩している．Standard ML の型体系のほとんどを Twelf で証明した [4] や OCaml の操作的意味論に注目した [6] が印象的である．しかし、OCaml の型システムは他言語と異なる機能を多くもっており，その形式的な証明が困難を伴う．たとえば，オブジェクト型や多相ヴァリアント型は構造的多相性と再起型を必要としており，今まで形式化された型体系より扱いが複雑と思われる．他にも副作用との共存を意識した緩和された値多相性，多相メソッドのための第一級多相性，ラベル付引数，構造や名前による部分型など，型体系の拡張が盛りだくさん．それぞれの拡張について実行の安全性を保証する型健全性だけでなく，複雑な型推論アルゴリズムの健全性と完全性も焦点になる．

現在のコンパイラについて証明を行うのは不可能に近いので，この研究では完全に証明された新しい参照実装の作成を目標とする．今回は再帰型を含んだ構造的多相性を追加した Core ML に対して完全に証明されたインタープリタを Coq で実装した．証明の対象は型体系および評価関数の型健全性，操作的意味論に対する評価関数の完全性，型推論アルゴリズムの健全性と完全性（主要性）である．プログラム抽出によって得られる実装が完全に証明されていることになる．

A Certified Interpreter for ML with Structural Polymorphism

ジャックガリグ，名古屋大学多元数理科学研究科，Graduate School of Mathematical Sciences, Nagoya University,

2 構造的多相性

多相ヴァリアントやオブジェクト型に利用される構造的多相性は型の幅の多相性と相互再帰型に基づいている．その形式化 [3] は再帰カインド環境を用いて行われた．基本的な定義を以下に示す．

$$
\begin{array}{lcll}
\tau & ::= & \alpha \mid \tau_1 \to \tau_2 & \text{型} \\
\kappa & ::= & \bullet \mid (C, \{l_1 \mapsto \tau_1, \ldots, l_n \mapsto \tau_n\}) & \text{カインド} \\
\mathrm{K} & ::= & \alpha_1 :: \kappa_1, \ldots, \alpha_n :: \kappa_n & \text{カインド環境} \\
\sigma & ::= & \forall \bar{\alpha}.K \triangleright \tau & \text{多相型}
\end{array}
$$

型は型変数または関数型である．しかし，型変数の抽象性はカインドによって具体化できる．カインド環境で具体的なカインドと関連づけられた型変数はレコードやヴァリアントのような構造型を表現する．局所制約 C とラベルから型への写像の対がその型の性質を示す．逆に，$\bullet$ は通常の（抽象的な）型変数を表す．カインド環境で関連づけられた型がカインドの中にも出現できるので，（カインドを通した）相互再帰型が自然に定義できる．

カインド環境は 2 カ所で使われる．多相型では

$$\textsc{Variable}\quad \frac{\mathrm{K}, \mathrm{K}_0 \vdash \theta : \mathrm{K} \quad \mathrm{dom}(\theta) \subset B}{\mathrm{K}; \Gamma, x : \forall B.\mathrm{K}_0 \triangleright \tau \vdash x : \theta(\tau)}$$

$$\textsc{Abstraction}\quad \frac{\mathrm{K}; \Gamma, x : \tau \vdash e : \tau'}{\mathrm{K}; \Gamma \vdash \lambda x.e : \tau \to \tau'}$$

$$\textsc{Application}\quad \frac{\mathrm{K}; \Gamma \vdash e_1 : \tau \to \tau' \quad \mathrm{K}; \Gamma \vdash e_2 : \tau}{\mathrm{K}; \Gamma \vdash e_1\ e_2 : \tau'}$$

$$\textsc{Generalize}\quad \frac{\mathrm{K}; \Gamma \vdash e : \tau \quad B = \mathrm{FV}_\mathrm{K}(\tau) \setminus \mathrm{FV}_\mathrm{K}(\Gamma)}{\mathrm{K}|_{\overline{B}}; \Gamma \vdash e : \forall B.\mathrm{K}|_B \triangleright \tau}$$

$$\textsc{Let}\quad \frac{\mathrm{K}; \Gamma \vdash e_1 : \sigma \quad \mathrm{K}; \Gamma, x : \sigma \vdash e_2 : \tau}{\mathrm{K}; \Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\textsc{Constant}\quad \frac{\mathrm{K}_0 \vdash \theta : \mathrm{K} \quad \mathrm{Tconst}(c) = \mathrm{K}_0 \triangleright \tau}{\mathrm{K}; \Gamma \vdash c : \theta(\tau)}$$

図 1　型付け規則

$$\textsc{Variable}\quad \frac{\mathrm{K} \vdash \bar{\tau} :: \bar{\kappa}^{\bar{\tau}}}{\mathrm{K}; \Gamma, x : \bar{\kappa} \triangleright \tau_1 \vdash x : \tau_1^{\bar{\tau}}}$$

$$\textsc{Abstraction}\quad \frac{\forall x \notin L \qquad \mathrm{K}; \Gamma, x : \tau \vdash e^x : \tau'}{\mathrm{K}; \Gamma \vdash \lambda e : \tau \to \tau'}$$

$$\textsc{Application}\quad \frac{\mathrm{K}; \Gamma \vdash e_1 : \tau \to \tau' \quad \mathrm{K}; \Gamma \vdash e_2 : \tau}{\mathrm{K}; \Gamma \vdash e_1\ e_2 : \tau'}$$

$$\textsc{Generalize}\quad \frac{\forall \bar{\alpha} \notin L \quad \mathrm{K}, \bar{\alpha} :: \bar{\kappa}^{\bar{\alpha}}; \Gamma \vdash e : \tau^{\bar{\alpha}}}{\mathrm{K}; \Gamma \vdash e : \bar{\kappa} \triangleright \tau}$$

$$\textsc{Let}\quad \frac{\mathrm{K}; \Gamma \vdash e_1 : \sigma \quad \forall x \notin L \quad \mathrm{K}; \Gamma, x : \sigma \vdash e_2^x : \tau}{\mathrm{K}; \Gamma \vdash \mathsf{let}\ e_1\ \mathsf{in}\ e_2 : \tau}$$

$$\textsc{Constant}\quad \frac{\mathrm{K} \vdash \bar{\tau} :: \bar{\kappa}^{\bar{\tau}} \quad \mathrm{Tconst}(c) = \bar{\kappa} \triangleright \tau_0}{\mathrm{K}; \Gamma \vdash c : \tau_0^{\bar{\tau}}}$$

図 2　局所匿名と補有限量化による型付け規則

抽象化された型変数のカインドを与える．型判定 $\mathrm{K}; \Gamma \vdash e : \tau$ では Γ や τ の型変数は K で関連づけられる．型判定の導出規則は図 1 に示した．$\mathrm{K} \vdash \theta : \mathrm{K}'$ は θ が K と K' の間でカインドを保つという意味である．厳密には α が K で具体的なカインドをもつ ($\alpha :: \kappa \in \mathrm{K},\ \kappa \neq \bullet$) ならば，$\theta(\alpha) = \alpha'$ も変数であり，K' の中でより具体化したカインドを持つ ($\alpha' :: \kappa' \in \mathrm{K}'$ かつ $\kappa' \models \theta(\kappa)$)．また，Generalize ではカインド環境をたどって自由な型変数を決め，結果として一般化される変数を用いてカインド環境を二分割する．細かい説明は [3] を参照されたい．

3　型健全性

Coq による証明の第一ステップは，上記の型体系の型健全性の証明である．その基礎として Aydemir 等 [1] の局所匿名 (locally nameless) な変数および補有限な量化 (cofinite quantification) による Core ML の型健全性を利用した．その証明の特徴として，項の中だけに De Bruijn 添字を使い，変数の衝突条件には抽象的な有限集合を使う．

このスタイルに合わせた型付け規則を図 2 に示した．$\bar{\tau}$ や $\bar{\kappa}$ は型やカインドの列を表している．$\bar{\alpha}$ と書いた場合は，さらに列の中の変数が二つずつ異なることが求められる．多相型は $\bar{\kappa} \triangleright \tau$ として表現され，$\bar{\kappa}$ の長さは抽象化される変数の数である．$\tau_1^{\bar{\tau}}$ は τ_1 の中の De Bruijn 添字を $\bar{\tau}$ の型に置き換えたものを表す．$\bar{\kappa}^{\bar{\tau}}$ は同様に列 $\bar{\kappa}$ の全てのカインドの中の添字を置き換えたものである．e^x は添字 1 だけを x に置き換えている．$\mathrm{K} \vdash \tau :: \kappa$ は $\kappa = \bullet$ または $\tau = \alpha$ で $\alpha :: \kappa' \in \mathrm{K}$ かつ $\kappa' \models \kappa$ のときに成り立つ．$\mathrm{K} \vdash \bar{\tau} :: \bar{\kappa}$ と書くと，$\bar{\tau}$ と $\bar{\kappa}$ の同位置にある型とカインドについて前記の条件が満され，θ がカインドを保つための条件と同等になる．

$\forall x \notin L$ や $\forall \bar{\alpha} \notin L$ は補有限量化に当たる．一見，図 1 と全く違うように見えるが，L の中身を具体化させると一致する．たとえば，$\forall x \notin L$ の L を Γ の定義域にすると変数の衝突が防げる．また Generalize の中の L を $\mathrm{dom}(\mathrm{K}) \cup \mathrm{FV}_K(\Gamma)$ にすればいい．直感的には理解しにくいが，量化の条件を陰的に表現するとても賢い方法である．導出の構造を変えたときにそれぞれの L を増やすだけで済むので，名前の付け替えに関する多くの証明が省けるのが最大の利点である．

このように，既にある証明を元にしたのが大きな助けになった．しかし様々な所で工夫が必要だった．構造的多相性の体系に関して，代入におけるカインドの条件を局所匿名に合わせた形に変える必要があった．また，証明を拡張するときに様々な新しい定義を導入することになった．たとえば，[1] の証明では変数の

KIND GC
$$\frac{\mathrm{K}, \mathrm{K}'; \Gamma \vdash e : \tau \quad \mathrm{FV}_{\mathrm{K}}(E, \tau) \cap \mathrm{dom}(\mathrm{K}') = \emptyset}{\mathrm{K}; \Gamma \vdash e : \tau}$$
CO-FINITE KIND GC
$$\frac{\forall \bar{\alpha} \notin L \quad \mathrm{K}, \bar{\alpha} :: \bar{\kappa}^{\bar{\alpha}}; \Gamma \vdash e : \tau}{\mathrm{K}; \Gamma \vdash e : \tau}$$

図 3 カインドの隠蔽

衝突は全て $\bar{\alpha} \notin L$ という形の条件で表現できていたが，もっと一般的な $L_1 \cap L_2 = \emptyset$ という形が必要になり，完全に自動化されていた衝突に関する証明が半自動になってしまった．新しい tactic を開発しても，この種の簿記的な証明を完全になくすことはできなかった．

さらに，定数および δ 規則を扱うための枠組みを追加した．これらは構造型に意味を与えるために必要だった．全体で，型健全性の証明の長さは 1000 行から 2000 行以上に倍増したが，ほとんどの変更は簡単だった．証明の途中で再帰型の存在がそれほど大きな問題にはならなかったが，一般的に再帰型を含む型健全性の証明は珍しく，私が知っているのは [4] だけである．

局所制約による構造的多相性の体系は元々フレームワークという形で定義されている．制約に関する定義を変えるだけで様々なヴァリアントやオブジェクト型が扱える．これを Coq のファンクターを使って表現した．期待通りの定義ができたが、欠点がある．カインドの定義が制約の定義に依存したり，制約に関する証明が型判定の定義に依存したりするので，ファンクターを入れ子にしなければならず，フレームワークの制約領域への適用は「対談」という形をとってしまう．この適用を多相ヴァリアントの制約領域について行った．

フレームワークおよび制約領域の証明では，補有限量化が功を奏し，型や項の中の名前の付け替えが不要だった．上でも触れたように，GENERALIZE の定義をより簡単にするという副作用もあった．元の体系では自由な型変数の定義がかなり複雑だが，補有限量化を使うとそんな定義が要らなくなる．ただ一つの難点は，名前の付け替えが要らないと思わせるあまり，それが本当に必要になったときに証明の方針が間違えやすい．具体的には，モジュラリティのために元の体系に結論に現れないカインドを削除する規則を追加しようと思った．図 3 にその規則の通常版と補有限版を書いた．しかし，この規則は構文主導ではないので，以下のような反転補題が必要になる．

$\forall \mathrm{K}\Gamma e\tau,\ \mathrm{K}; \Gamma \vdash_{GC} e : \tau \Rightarrow \exists \mathrm{K}',\ \mathrm{K}, \mathrm{K}'; \Gamma \vdash e : \tau$

拡張した体系で型付け可能な項はカインド環境を増やすだけで元の体系で型付け可能である．この補題は直感的だが，それを補有限版の体系で証明しようとするとはまってしまう．量化子の順序を変えなければならないので，項と型変数両方の名前を付け替えなければならない．そのせいでこの補題の証明に（付け替え補題を含めて）1000 行もかかってしまった．問題の原因が分かれば，もっと楽な方法も見つかり，KIND GC を完全になくすのではなく，ABSTRACTION と GENERALIZE の直前に移動させる標準化補題は 100 行程度で証明でき，ほとんどの用途には十分である．

4 型推論

局所制約の最大の目的はカインドの単一化を可能にし，型推論アルゴリズムを複雑な制約解消アルゴリズムに依存させないことである．当然ながら，カインドを扱うために単一化を拡張しなければならないが，基本的な考え方が変わらない．

単一化アルゴリズムの検証は古くから研究され，1985 年には既に Paulson が証明を与えている [7]．ここでは単純にアルゴリズムを Coq の関数として書き，部分的正当性および完全性を証明した．カインドを扱わなければならないので，証明は 1900 行近くあるが，大きな障害はなかった．基本的にアルゴリズムに沿った証明だが，二つの工夫がある．まず，代入の関係として通常の「θ が θ' より一般的である」($\exists \theta_1,\ \theta' = \theta_1 \circ \theta$) ではなく，それより少し強い「$\theta'$ が θ を拡張する」($\forall \alpha,\ \theta'(\theta(\alpha)) = \theta'(\alpha)$) という性質を使った．以降は $\theta' \sqsubseteq \theta$ と書く．θ が冪等のときに両者が同値であることが簡単に示せるが，後者は書き換えという形で簡単に使える．また，成功した場合のアルゴリズムに対する帰納法を定義し，部分的正当性の証明が短くできた．単一化は一階の項だけを扱っているので，ここでは De Bruijn 添字は影響しない．

$[\bar{\alpha}]\tau = \tau_*$ where $\tau_*^{\bar{\alpha}} = \tau$ and $\mathrm{FV}(\tau_*) \cap \bar{\alpha} = \emptyset$
$[\bar{\alpha}](\bar{\kappa} \triangleright \tau) = ([\bar{\alpha}]\bar{\kappa} \triangleright [\bar{\alpha}]\tau)$
$\mathrm{generalize}(\mathrm{K}, \Gamma, L, \tau) =$
  let $A = \mathrm{FV}_\mathrm{K}(\Gamma)$ and $B = \mathrm{FV}_\mathrm{K}(\tau)$ in
  let $\mathrm{K}' = \mathrm{K}|_{\overline{A}}$ in
  let $\bar{\alpha} :: \bar{\kappa} = \mathrm{K}'|_B$ in
  let $\bar{\alpha}' = B \setminus (A \cup \bar{\alpha})$ in
  let $\bar{\kappa}' = \mathrm{map}(\lambda\alpha.\bullet)\ \bar{\alpha}'$ in
  $((\mathrm{K}|_A, \mathrm{K}'|_L), [\bar{\alpha}\bar{\alpha}'](\bar{\kappa}\bar{\kappa}' \triangleright \tau))$
$\mathrm{typinf}(\mathrm{K}, \Gamma, \mathsf{let}\ e_1\ \mathsf{in}\ e_2, \tau, \theta, L) =$
  let $\alpha = \mathrm{fresh}(L)$ in
  match $\mathrm{typinf}(\mathrm{K}, \Gamma, e_1, \alpha, \theta, L \cup \{\alpha\})$ with
  | $\langle \mathrm{K}', \theta', L' \rangle \Rightarrow$
    let $\mathrm{K}_1 = \theta'(\mathrm{K}')$ and $\Gamma_1 = \theta'(\Gamma)$ in
    let $L_1 = \theta'(\mathrm{dom}(\mathrm{K}))$ and $\tau_1 = \theta'(\tau')$ in
    let $(\mathrm{K}_A, \sigma) = \mathrm{generalize}(\mathrm{K}_1, \Gamma_1, L_1, \tau_1)$ in
    let $x = \mathrm{fresh}(\mathrm{dom}(E) \cup \mathrm{FV}(e_1) \cup \mathrm{FV}(e_2))$ in
    $\mathrm{typinf}(\mathrm{K}_A, (\Gamma, x : \sigma), e_2^x, \tau, \theta', L')$
  | $\langle\rangle \Rightarrow \langle\rangle$
  end

図 4　型推論アルゴリズム

次のステップは型推論である．これも Core ML の場合の証明は既にある [5] [2]．しかし等価再帰型を含めたものは初めてと思われる．健全性と完全性（主要性）の両方が大変で，証明は併せて 3400 行に及ぶ．ここではアルゴリズムの複雑さと同時に守らなければ条件の多さが原因と思われる．特に一般化の正しいやり方が複雑だった．通常の ML では型の自由な型変数から環境で自由な型変数を引くだけでよいが、カインド環境があると複数のステップが追加される．まず，自由な型変数のカインド環境による閉包をとる．次に，カインド環境を一般化される部分と一般化されない部分に分割する．最後に，偶然な変数名の一致に対応するために一般化されるカインド環境の一部を一般化されない方に残さなければならない．generalize および typinf の let の場合を図 4 に示した．$[\bar{\alpha}]\tau$ は型 τ の中に型変数 $\bar{\alpha}$ を De Brujin 添字に置き換えたものを表している．

条件が多いせいで，帰納法で証明される健全性と主要性の定義が長くなる．図 5 の中で構造的な条件だけを省いた．$\mathrm{K} \vdash \Gamma_1 \leq \Gamma$ は Γ の中の型が Γ_1 の型のインスタンスであること (Γ_1 がより一般的である)

健全性
$\mathrm{typinf}(\mathrm{K}, \Gamma, \mathrm{e}, \tau, \theta, \mathrm{L}) = \langle \mathrm{K}', \theta', \mathrm{L}' \rangle \rightarrow$
$\mathrm{dom}(\theta) \cap \mathrm{dom}(\mathrm{K}) = \emptyset \rightarrow$
$\mathrm{FV}(\theta, \mathrm{K}, \Gamma, \tau) \subset L \rightarrow$
$\theta' \sqsubseteq \theta \wedge \mathrm{dom}(\theta') \cap \mathrm{dom}(\mathrm{K}') = \emptyset \wedge$
$\mathrm{K} \vdash \theta' : \theta'(\mathrm{K}') \wedge \theta'(\mathrm{K}'); \theta'(\Gamma) \vdash e : \theta'(\tau) \wedge$
$\mathrm{FV}(\theta', \mathrm{K}', \Gamma) \cup L \subset L'.$
主要性
$\mathrm{K}; \Gamma \vdash e : \theta(\tau) \rightarrow$
$\mathrm{K} \vdash \theta(\Gamma_1) \leq \Gamma \rightarrow \theta \sqsubseteq \theta_1 \rightarrow \mathrm{K}_1 \vdash \theta : \mathrm{K} \rightarrow$
$\mathrm{dom}(\theta_1) \cap \mathrm{dom}(\mathrm{K}_1) = \emptyset \rightarrow$
$\mathrm{dom}(\theta) \cup \mathrm{FV}(\theta_1, \mathrm{K}_1, \Gamma_1, \tau) \subset L \rightarrow$
$\exists \mathrm{K}'\theta' L',\ \mathrm{typinf}(\mathrm{K}_1, \Gamma_1, e, \tau, \theta_1, L) = \langle \mathrm{K}', \theta', L' \rangle \wedge$
$\exists \theta'',\ \mathrm{dom}(\theta'') \subset L' \setminus L\ \wedge \theta\theta'' \sqsubseteq \theta' \wedge \mathrm{K}' \vdash \theta\theta'' : \mathrm{K}.$

図 5　型推論に関する定理

を意味する．インスタンスの具体的な定義もカインドがあるせいで通常の定義より複雑になる．

$$\mathrm{K} \vdash \bar{\kappa_1} \triangleright \tau_1 \leq \bar{\kappa} \triangleright \tau \stackrel{\mathrm{def}}{=} \forall \bar{\alpha}, \mathrm{dom}(\mathrm{K}) \cap \bar{\alpha} = \emptyset \rightarrow \exists \bar{\tau},\ \mathrm{K}, \bar{\alpha} :: \bar{\kappa}^{\bar{\alpha}} \vdash \bar{\tau} :: \bar{\kappa_1}^{\bar{\tau}} \wedge \tau_1^{\bar{\tau}} = \tau^{\bar{\alpha}}.$$

代入による元の定義が分かりやすいかも知れない．

$$\mathrm{K} \vdash \forall \bar{\alpha_1}.\mathrm{K}_1 \triangleright \tau_1 \leq \forall \bar{\alpha}_2.\mathrm{K}_2 \triangleright \tau_2 \stackrel{\mathrm{def}}{=} \exists \theta,\ \mathrm{dom}(\theta) \subset \bar{\alpha_1} \wedge \mathrm{K}, \mathrm{K}_1 \vdash \theta : \mathrm{K}, \mathrm{K}_2 \wedge \theta(\tau_1) = \tau_2.$$

証明を長くするもう一つの原因は，型判定の導出木を新しく作らなければならないので，補有限量化は利用できるものではなく，こちらが保証する条件になり，多くの名前の付け替えが必要になる．健全性だけでなく，主要性についても，新しい型判定と入力の型判定が大きく違うので，同じ事がいえる．

5　インタープリタ

前記の型健全性はプログラムを所定の書き換え規則で書き換えても型の整合性が保たれることを保証している．しかし，プログラミング言語の処理系では，プログラムを書き換えるのではなく，抽象機械などでそれを解釈実行することが普通である．今回は SECD のような抽象機械を定義し，評価の各ステップで機械の状態をプログラムに逆変換すれば書き換えられたプログラムと同様に型付けできる形になっていることを証明した．これによってプログラムの実行時に型エラーが起きないことが保証され，もしも実行が停止すれば，その時点での結果が正しい型の定数か関数閉

包であることも保証される．プログラムと抽象機械の状態の関係を定義したら，証明は難しくなかった．項が De Bruijn 添字を既に使っているので，コンパイルなしにスタックに基いた抽象機械で実行できるのが便利だった．

さらに，書き換えのよる評価が正規形に辿り着いた場合，抽象機械による評価の結果が同じ正規形を表していることを証明した．こちらは途中で計算の相模倣のようなものが必要になって，意外と難しかった．

6 プログラムの抽出

型検証器とインタプリタの両方が Coq のプログラム抽出機能によって OCaml のプログラムになる．これにより，OCaml の型システムの一部について完全に検証された実装が与えられた．ただし，パーザや入出力ループがまだないので，抽象構文で書かれたプログラムを型推論・実行することしかできない．また，Coq の関数が明示的な停止性を持たなければならないので，実行するステップ数を書かなければならない．(実際には，OCaml で再帰的なデータが書けるので，自然数の無限大に当る値が作れ，ステップ数を省略することは可能だが，Coq の保証範囲から出てしまう．)

停止性の問題は単一化アルゴリズムや型推論アルゴリズムの書き方にも影響する．特に単一化アルゴリズムの停止性は自明ではないので，元々関数に計算するステップ数を引数として渡し，それが 0 になったら失敗するようにした．後に一定以上の数を渡すと 0 にならないことを完全性の中で証明することになる．この方法は手軽で良いが，抽出したコードの中にこの余計な引数に関する計算が残ってしまう．最初はその引数を大き過ぎる数にして，実際に単一化が全く使えなかったこともある．もっと賢い引数の選び方でそれを避けられるが，できることならこの引数を完全になくしたい．Coq ではよく知られている技があり，引数を命題の世界 (Prop) に移してしまうと抽出時に消える．そのためにかなり激しく依存型を使う必要があるが，その反面で完全性の証明が簡単になる利点があり，それを単一化アルゴリズムに適用しても証明の長さが変わらなった．しかし，そのために単一化を使う型推論アルゴリズムも変更しなければならなくて，そちらの証明は結果的に 10%長くなった．ちなみに，関数定義の中で依存型を使うときに Program 環境を使うことが多いが，それで作った関数に対して書き換えが困難になるため，定義には Program 環境を利用しなかった．

7 まとめ

構造的多相性をもった言語の型推論およびインタープリタを完全に検証することに成功した．プログラム抽出によってそれらを実行することも可能である．今後は OCaml の型システムの他の機能を順次追加することになるが，今回経験した困難から，それが簡単に行くかどうかまだ断定できない．

Coq の証明および抽出されたコードが以下の URL から入手できる:

`http://www.math.nagoya-u.ac.jp/~garrigue/papers/#certint0908`

参 考 文 献

[1] Aydemir, B., Charguéraud, A., Pierce, B. C., Pollack, R., and Weirich, S.: Engineering Formal Metatheory, *Proc. ACM Symposium on Principles of Programming Languages*, 2008, pp. 3–15. Proofs at `http://www.chargueraud.org/arthur/research/2007/binders/`.

[2] Dubois, C. and Ménissier-Morain, V.: Certification of a Type Inference Tool for ML: Damas-Milner within Coq, *Journal of Automated Reasoning*, Vol. 23, No. 3(1999), pp. 319–346.

[3] Garrigue, J.: Simple type inference for structural polymorphism, *The Ninth International Workshop on Foundations of Object-Oriented Languages*, Portland, Oregon, 2002.

[4] Lee, D. K., Crary, K., and Harper, R.: Towards a Mechanized Metatheory of Standard ML, *Proc. ACM Symposium on Principles of Programming Languages*, January 2007, pp. 173–184.

[5] Naraschewski, W. and Nipkow, T.: Type Inference Verified: Algorithm W in Isabelle/HOL, *Journal of Automated Reasoning*, Vol. 23(1999), pp. 299–318.

[6] Owens, S.: A Sound Semantics for OCaml light, *Proc. European Symposium on Programming*, LNCS, Vol. 4960, April 2008, pp. 1–15.

[7] Paulson, L.: Verifying the Unification Algorithm in LCF, *Science of Computer Programming*, Vol. 5(1985), pp. 143–169.