修 士 論 文

型代入を遅延する最適化型推論アルゴリズム

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

上野 雄大

2006 年 3 月

修 士 論 文

型代入を遅延する最適化型推論アルゴリズム

指導教員　大堀淳 教授

審査委員主査　大堀淳 教授
審査委員　日比野靖 教授
審査委員　小川瑞史 助教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

410015 上野 雄大

提出年月: 2006 年 2 月

概 要

型推論は型付き関数型言語のコンパイルステップの中でも複雑でかつ時間を要する処理であるにもかかわらず，その最適化方式や最適化による効果などはあまり研究されていない．本稿では，従来の型推論アルゴリズム $\mathcal{W}$ と比較して実用上より効率的であると期待でき，かつ宣言的に定義される新たな型推論アルゴリズム $\mathcal{DW}$ を提案し，その正しさを示すとともに，$\mathcal{DW}$ を実用的なコンパイラへ採用する上での課題に関して議論する．さらに，そのアルゴリズムを実際のコンパイラ上に実装し，その実用性を示す．

目 次

第1章　序論

1.1　型推論

型推論機構は，Standard ML や Haskell，ObjectiveCaml などの近代的な関数型言語の大きな特徴のひとつである．これは，プログラムが持つ最も一般的な型を自動的に推論する機構である．この機構によって，コンパイラは明示的に型付けされていないプログラムの最も一般的な型を自動的に推論する．例えば，

$$\lambda x.x$$

のような式が与えられた場合，コンパイラは型推論機構によってこの式の型を推論し，その結果この式に対し $t \to t$ という型を付ける．この型は，ある型 t の値を受けとり型 t の値を返す関数を表す．t にどのような型が入るかは，この式の前後で，この関数に対してどのような引数を渡しているかで決定される．もしコンパイラが与えられた式に型を付けることができなければ，型エラーを表示してコンパイルを中止する．型推論機構によって，ユーザーは煩雑な型宣言構文を一切書くことなく，型によるプログラムの検査などの型付き言語の恩恵を享受することができる．

また，プログラム中に明示的に型の指定をしないため，複数の型を持つ汎用的なプログラムの記述が可能である．特に，近代の関数型言語のコンパイラの多くは多相型のための変数束縛を許す多相型 let 構文と多相型を推論できる多相型型推論機構を備えており，様々な型のデータを受け取ることができる関数を明示的な宣言をせずに記述することができる．例えば，

$$\mathsf{let}\ f = \lambda x.x$$

という式が与えられた場合，f の型は $\forall t.t \to t$ と推論される．このような型を多相型といい，多相型を持つ関数を多相関数という．f の出現のたびに f の型として新たな多相型の例が生成され，プログラムの任意の場所で，任意の型のデータに対して関数 f を使用することができる．

1.2　背景と目的

多相型型推論機構が実践的に非常に有用であることは広く知られており，様々な言語の処理系が型推論アルゴリズムを実装している．しかし一方で，型推論アルゴリズムは関数

型言語のコンパイラフロントエンドの中でも最も複雑かつ時間のかかる処理のひとつであり，コンパイラの複雑化やコンパイル時間の増大を招いている．宣言的かつ効率的な型推論アルゴリズムの開発は，多相型型推論機構を装備した次世代プログラミング言語の設計と実装にとって重要な課題である．この重要性にもかかわらず，その最適化方式や最適化による効果などはあまり研究されていない．

一方，多相型 let 構文が存在するために，多相型を持つ関数型言語の型推論問題は DEXPTIME 完全であることがすでに示されており [8, 5, 6]，漸近的な動作を考えた場合，アルゴリズム論の意味において効率的な型推論アルゴリズムの構築は不可能である．このような理論的な限界は存在しているものの，コンパイル時間の短縮のため，現実には実用上より効率的な型推論アルゴリズムが求められている．現在，多くの関数型言語のコンパイラで採用されている型推論アルゴリズムは，Milner の先駆的な研究 [9] によって与えられたアルゴリズムに実用上の効率化のための改良を加えたものとなっている．しかし，この拡張は手続き的な機能を用いてアドホックに行われているため，アルゴリズムの実装の可読性は低く，またその手法は広く知られているものの，その拡張アルゴリズムの性質は明らかになっていない．

本研究の一般的な目的は，高度な機能を含む最先端の関数型プログラミング言語のコンパイラの高信頼かつ効率的な実装の基礎として，実用上より効率的で宣言的記述が可能な型推論アルゴリズムとその実装技術を構築することである．本稿では，その第一歩として新しい型推論アルゴリズムを提案し，その正しさを示す．また提案するアルゴリズムを実装し，その実現可能性を示すとともに，その実用上の性能を評価する．最後に，アルゴリズムの拡張可能性や更なる効率化のための課題などを議論する．

1.3 構成

次章以降の本稿の構成は以下の通りである．第 2 章では，従来の型推論アルゴリズムの問題点を指摘し，より効率的なアルゴリズムを構築する上での基本的なアイディアを概説する．第 3 章では，型推論アルゴリズムを与え，その型健全性を証明する．第 4 章では，提案した型推論アルゴリズムの実装の概要について述べ，アルゴリズムが実際のコンパイラ上に実現可能であることを示し，さらにその実装を用いてアルゴリズムの定量的な評価を行う．

第 5 章では，提案する型推論アルゴリズムと従来のアドホックな効率化手法や関連研究との比較・検討を行う．最後に第 6 章では，結論とアルゴリズムの種々の拡張の可能性を含む今後の課題に関して議論する．

第2章　型推論アルゴリズム$\mathcal{W}$

本章では，今日の多くのコンパイラの実装の基礎となっている Milner の型推論アルゴリズム $\mathcal{W}$ の概要を述べ，その効率上の問題点を指摘する．さらに，それらを解決するための基本的なアイディアについて概説し，効率的な型推論アルゴリズムを構築するための基礎を与える．

2.1　式と型の定義

最初に，議論を明確にするために，本稿で分析の対象とする式と型，および型システムを定義する．

本稿で扱う式の集合は，Milner の型推論アルゴリズム [9] に合わせて，以下の BNF 文法で定義する．

$$e ::= c^b \mid x \mid \lambda x.e \mid \mathsf{fix}\ x.e \mid e\ e \mid \mathsf{let}\ x = e\ \mathsf{in}\ e$$

ここで，c^b は基底型 b の定数，x は変数，$\lambda x.e$ はラムダ抽象，$\mathsf{fix}\ x.e$ は $\lambda x.e$ の最小不動点，$\mathsf{let}\ x = e\ \mathsf{in}\ e$ は多相型 let 構文である．

単相型（τ）と多相型（σ）の集合は以下の文法で定義される．

$$\tau ::= b \mid t \mid \tau \to \tau$$
$$\sigma ::= \tau \mid \forall \overline{t}.\tau$$

ここで，b は基底型，t は型変数，$\overline{t}$ は型変数の集合を表す．

型システムは，一般的な ML の型導出システムに合わせて，$\Gamma \rhd e : \sigma$ の形の型判定導出システムとして定義する．図 2.1 に型導出規則の集合を与える．

この型システムでは，多相型 let 構文に対する型導出規則において，型環境 Γ や多相化の対象となる型 τ_1 に自由な出現を持たない型変数が束縛されていても良いように，束縛変数の集合に関する条件を緩和している．この点は一般的な ML 型の型導出システムと異なる．しかし，このように条件を緩和しても，多相型内に全く出現を持たない変数を束縛することを許すだけであり，型システムの整合性には何ら問題はない．

$$\text{(const)}\quad \Gamma \rhd c^b : b$$

$$\text{(var)}\quad \Gamma\{x:\sigma\} \rhd x : \tau \qquad (\tau < \sigma)$$

$$\text{(abs)}\quad \frac{\Gamma\{x:\tau_1\} \rhd e : \tau_2}{\Gamma \rhd \lambda x.e : \tau_1 \to \tau_2}$$

$$\text{(fix)}\quad \frac{\Gamma\{x:\tau\} \rhd e : \tau}{\Gamma \rhd \mathsf{fix}\ x.e : \tau}$$

$$\text{(app)}\quad \frac{\Gamma \rhd e_1 : \tau_1 \to \tau_2 \qquad \Gamma \rhd e_2 : \tau_1}{\Gamma \rhd e_1\ e_2 : \tau_2}$$

$$\text{(let)}\quad \frac{\Gamma \rhd e_1 : \tau_1 \qquad \Gamma\{x : \forall \overline{t}.\tau_1\} \rhd e_2 : \tau_2}{\Gamma \rhd \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2} \qquad (\overline{t} \cap \mathrm{FTV}(\Gamma) = \emptyset)$$

図 2.1: 対象とする型システム

2.2 型推論アルゴリズム $\mathcal{W}$

第 1 章で述べた通り，現在多くのコンパイラに実装されている型推論アルゴリズムは Milner の型推論アルゴリズム $\mathcal{W}$ を基礎とする．本節では，$\mathcal{W}$ の概要を述べる．

$\mathcal{W}$ は，Robinson の単一化アルゴリズム [15] を基礎として構築されている．単一化アルゴリズム $\mathcal{U}$ は，1 つの型のペアを受け取り型代入を返す関数として定義される．$\mathcal{U}$ は以下のような条件を満たす．

- $\mathcal{U}(\tau_1, \tau_2)$ が成功したならば，τ_1 と τ_2 を単一化するような型代入を返す．
- もし型 τ_1 と τ_2 の間に単一化が存在するならば，$\mathcal{U}(\tau_1, \tau_2)$ は成功し，τ_1 と τ_2 の最も一般的な単一化を返す．
- $\mathcal{U}(\tau_1, \tau_2)$ が返す型代入は，τ_1 および τ_2 に出現を持つ型変数のみに影響する．

$\mathcal{W}$ は，型環境 Γ と式 e を受け取り，型変数への代入 S と型 τ を返す関数として定義される．図 2.2 にその定義を示す．

$\mathcal{W}$ は，各部分式の型が満たすべき制約を型変数を用いて型等式の集合として構築し，そのような制約を満たす最も一般的な解を単一化アルゴリズムによって計算することで式 e の型を推論する．もし型環境 Γ の下で式 e が型を持つならば，$\mathcal{W}$ はその最も一般的な型判定 $\Gamma \rhd e : \sigma$ を返し，もし e が型を持たなければエラーを報告する．

$$\mathcal{W}(\Gamma\{x:\tau\}, x) = (\emptyset, \tau)$$

$$\mathcal{W}(\Gamma\{x:\forall(t_1,\ldots,t_n).\tau\}, x) = (\emptyset, [t'_1/t_1,\ldots,t'_n/t_n]\tau) \quad (t'_1,\ldots,t'_n \text{ fresh})$$

$$\begin{aligned}\mathcal{W}(\Gamma, \lambda x.e) = \text{let } &(S_1,\tau_1) = \mathcal{W}(\Gamma\{x:t\}, e) \quad (t \text{ fresh})\\ &\text{in } (S_1, S_1(t)\to\tau_1)\end{aligned}$$

$$\begin{aligned}\mathcal{W}(\Gamma, \mathsf{fix}\ x.e) = \text{let } &(S_1,\tau_1) = \mathcal{W}(\Gamma\{x:t\}, e) \quad (t \text{ fresh})\\ &S_2 = \mathcal{U}(\{(S_1(t),\tau_1)\})\\ &\text{in } (S_2\circ S_1, S_2\circ S_1(t))\end{aligned}$$

$$\begin{aligned}\mathcal{W}(\Gamma, e_1\ e_2) = \text{let } &(S_1,\tau_1) = \mathcal{W}(\Gamma, e_1)\\ &(S_2,\tau_2) = \mathcal{W}(S_1(\Gamma), e_2)\\ &S_3 = \mathcal{U}(\{(S_2(\tau_1), \tau_2\to t)\}) \quad (t \text{ fresh})\\ &\text{in } (S_3\circ S_2\circ S_1, S_3(t))\end{aligned}$$

$$\begin{aligned}\mathcal{W}(\Gamma, \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) = \text{let } &(S_1,\tau_1) = \mathcal{W}(\Gamma, e_1)\\ &\overline{t} = \mathrm{FTV}(\tau_1)\setminus\mathrm{FTV}(\Gamma)\\ &(S_2,\tau_2) = \mathcal{W}(S_1(\Gamma)\{x:\forall\overline{t}.\tau_1\}, e_2)\\ &\text{in } (S_2\circ S_1, \tau_1)\end{aligned}$$

図 2.2: Milner の型推論アルゴリズム

2.3 $\mathcal{W}$ の効率上の問題点とその解決案

型推論アルゴリズム $\mathcal{W}$ の基本的な流れは，以下のように整理することができる．

1. 再帰的に自分自身を部分式に適用し，型と型代入を計算する．
2. 得られた型代入 S を以前の環境 Γ に適用し，型環境に含まれる型情報を更新する．
3. 更新された型環境のもとで，他の部分式の型推論を実行する．

アルゴリズムの効率の点からこの過程を分析すると，ステップ 2 に効率上の問題が見られる．多くの場合，型環境 Γ には型代入 S に無関係の環境が多数含まれているため，繰り返し行われる S の Γ への適用は，無駄な処理が多く非効率的である恐れがある．

このような変数への代入は関数型言語の評価規則にも見受けられる．関数適用式の評価において，実引数を束縛変数に代入する規則である．そこで，型推論における型代入の適用を関数型言語の評価に対応させて考えると，この事情は関数適用の評価を

$$(\lambda x_1 \ldots \lambda x_n.M)N_1\cdots N_n \Rightarrow [N_n/x_n]\cdots[N_1/x_1]M$$

のように構文的な代入を繰り返し行うことによって実現することに相当すると見なすことができる．しかし，実際の関数型言語のコンパイラでは，実行効率上の問題のためこのような実引数の代入は行わない．代わりに代入の効果を環境として保存し，その環境の下で M を評価するという戦略をとることによって実行効率の良い実装を実現している．

そこで，型推論においても，部分式に対して推論される型代入を現在の環境に適用した上で推論を実行するのではなく，型代入を型環境を評価する際に使用すべき明示的な環境として保存するようにアルゴリズムを再編成すれば，型代入を環境に繰り返し適用することを抑止でき，より効率の良いアルゴリズムが実現できると期待できる．

型推論アルゴリズム $\mathcal{W}$ が非効率的であるもう一つの要因は，多相型 let 構文の型推論において行われる $\mathrm{FTV}(\tau) \setminus \mathrm{FTV}(\Gamma)$ の計算である．これを計算するためには，多くの型を含んでいる型環境 Γ 全体のスキャンが必要となる．もし Γ から到着可能な型変数の集合を常に正確に把握することができるならば，Γ 全体のスキャンを多相型 let 構文の型推論の際に毎回実行するような無駄を省くことができるはずである．

多相型 let 構文は，一般的なプログラムでは関数や変数の束縛に数多く使用されるため，現実のプログラムの型推論では束縛変数の集合の計算は頻繁に行われる．多相型 let 構文の型推論を効率化することによって，実際のコンパイル処理における型推論に要する時間の大きな短縮が期待できる．

第3章　型代入を遅延する型推論アルゴリズム

本章では，前章で述べた従来の型推論アルゴリズム $\mathcal{W}$ の効率上の問題点に対する洞察に基づき，型推論アルゴリズム $\mathcal{W}$ を洗練したより効率的な型推論アルゴリズム $\mathcal{DW}$ を提案する．また，型推論アルゴリズム $\mathcal{DW}$ が対象とする言語の型システムに対して健全であることを証明する．

3.1　概要

従来の型推論アルゴリズム $\mathcal{W}$ は型環境 Γ と式 e を受け取り，Γ の下での e の型を推論して，型代入 S と e の型 τ を返すものであった．本章では，2.3 節での洞察に基づき，$\mathcal{W}$ を基礎として，実用上より効率的な型推論アルゴリズム $\mathcal{DW}$ を構築する．

具体的には，型推論アルゴリズムの引数として型環境 Γ に加え，Γ に対する型代入環境 S と，Γ から到着可能な型変数集合 Δ を導入し，これらの情報から新たな型代入 S'，新たに型環境から到着可能になった型変数の集合 Δ'，および式の型 τ を計算する

$$\mathcal{DW}(\Gamma, S, \Delta, e) = (S', \Delta', \tau)$$

という形のアルゴリズム $\mathcal{DW}$ を構築する．

$\mathcal{DW}$ では，推論の過程で新たな型代入が得られたとしても，それをただちに型環境や型に適用することはしない．代わりに型代入の効果を型代入環境に保存しておき，推論を進める上で実際に型代入が適用された型が必要になる時点まで型代入の適用を遅延する．

また，$\mathcal{DW}$ は，従来の $\mathcal{W}$ 同様，型の制約を満たす最も一般的な解を，型等式に対する最も一般的な単一化を計算することによって求めることで型推論問題を解決する．このために単一化アルゴリズムを必要とする．型代入を遅延するという $\mathcal{DW}$ の戦略を実現するためには，単一化アルゴリズムも型代入環境 S の下で型等式の単一化を計算するアルゴリズムへと洗練する必要がある．

本章では，このように洗練された単一化アルゴリズム $\mathcal{DU}$ と，それを用いた $\mathcal{DW}$ の 2 つのアルゴリズムの定義を与え，その正しさをそれぞれ証明する．

3.2 型代入

議論を明確にするために，アルゴリズムの定義に先立って，アルゴリズムで取り扱う型代入の構造を定義する．

任意の型を含む構造 X について，X に含まれる型変数の集合を $\mathrm{FTV}(X)$ と書く．型代入 S は，代入の対象となる型変数 t と，それに代入される型 τ のペア (t, τ) の集合とする．型代入 $S = \{(t_1, \tau_1), \ldots, (t_n, \tau_n)\}$ が以下の条件を全て満たすとき, S は**整形されている** *(well-formed)* という．

- $t_1, \ldots, t_n$ は相異なる.
- $t_1, \ldots, t_n$ は $\tau_1, \ldots, \tau_n$ に現れない.

また，型代入 S に含まれる代入の対象となる型変数の集合 $\{t_1, \ldots, t_n\}$ を $\mathrm{dom}(S)$ と書く．

型代入 S は，型変数から型への関数，またはその定義域を型変数から任意の型変数を含む構造に拡張した準同型写像とみなすことができる．S が整形されているとき，任意の型を含む構造 X について，X に含まれる自由な型変数を型代入 S によって対応付けられている型で置き換えた構造 X' を求める操作を $S(X) = X'$ と書く．さらに，任意の型代入 S_1，S_2 について，S_1 と S_2 の合成 $S_1 \circ S_2$ を，任意の型を含む構造 X に対して $S_1 \circ S_2(X) = S_1(S_2(X))$ と定義する．

型代入について，以下の補題が成立する．

補題 3.1. 任意の型代入 S_1，S_2 について，$\mathrm{dom}(S_1 \circ S_2) = \mathrm{dom}(S_1) \cup \mathrm{dom}(S_2)$.

補題 3.2. 任意の型代入 S_1，S_2，S_3，任意の型を含む構造 X について，$(S_1 \circ S_2) \circ S_3(X) = S_1 \circ (S_2 \circ S_3)(X)$.

補題 3.3. 任意の型代入 S，任意の型を含む構造 X について，S が整形されているならば，$\mathrm{FTV}(S(X)) \cup \mathrm{dom}(S) = \emptyset$ である．

補題 3.4. S を任意の型代入，τ を任意の型，t を任意の型変数，$S' = \{(t, \tau)\}$ とする．もし S が $S(\tau) = S(t)$ を満たすならば，任意の型 τ' について $S(S'(\tau)) = S(\tau')$ である．

補題 3.5. 任意の型環境 Γ，任意の式 e，任意の型 τ，任意の型代入 S について，$\Gamma \triangleright e : \tau$ ならば $S(\Gamma) \triangleright e : S(\tau)$ である．

補題 3.6. 任意の型代入 S，任意の型変数の集合 Δ_1，Δ_2 について，$\Delta_1 \subseteq \Delta_2$ ならば $\mathrm{FTV}(S(\Delta_1)) \subseteq \mathrm{FTV}(S(\Delta_2))$ である．

補題 3.7. 任意の型代入 S，型 τ について，$\mathrm{FTV}(S(\tau)) = \mathrm{FTV}(S(\mathrm{FTV}(\tau)))$.

各補題の証明は省略する．

$$
\begin{array}{ll}
\text{(u-i)} & (S, S', E \cup \{(\tau, \tau)\}) \Longrightarrow (S, S', E) \\
\text{(u-ii)} & (S, S', E \cup \{(\tau_{11} \to \tau_{12}, \tau_{21} \to \tau_{22})\}) \Longrightarrow (S, S', E \cup \{(\tau_{11}, \tau_{21}), (\tau_{12}, \tau_{22})\}) \\
\text{(u-iii)} & (S, S', E \cup \{(t, \tau)\}) \Longrightarrow (S, \{(t, S' \circ S(\tau))\} \cup (\{(t, S' \circ S(\tau))\})(S'), E) \\
 & \quad \text{if } t \notin \mathrm{dom}(S' \circ S),\ t \notin \mathrm{FTV}(S' \circ S(\tau)) \\
\text{(u-iv)} & (S, S', E \cup \{(t, \tau)\}) \Longrightarrow (S, S', E \cup \{(S' \circ S(t), \tau)\}) \\
 & \quad \text{if } t \in \mathrm{dom}(S' \circ S)
\end{array}
$$

図 3.1: 単一化アルゴリズム $\mathcal{DU}$ の変形規則

3.3 単一化アルゴリズム $\mathcal{DU}$

単一化アルゴリズム $\mathcal{DU}$ は，Gallier と Snyder の考え方 [3] に従い，型等式間の変形関係 $\Longrightarrow$ を通じて以下のような関数として定義される．

$$
\mathcal{DU}(S, E) = \begin{cases} S' & \text{if } (S, \emptyset, E) \overset{*}{\Longrightarrow} (S, S', \emptyset), \\ \mathit{failure} & \text{otherwise.} \end{cases}
$$

ここで，S および S' は型代入，E は型等式の集合である．$\mathcal{DU}$ は，任意の $(\tau_1, \tau_2) \in E$ について $S(\tau_1) = S(\tau_2)$ となるような型代入 S を返す．このような性質を満たす S を，E の単一化（unifier）と言う．

$\mathcal{DU}$ で用いている変形規則の定義を図 3.1 に示す．各変形規則は，必要に応じて型等式 E に含まれる型変数を型代入環境 S および新たに生成した型代入 S' で解決しながら，型等式 E の単一化を求めるステップを実現する．

変形規則の定義より，以下の補題が成立することは明らかである．

補題 3.8. $(S, S_1, E_1) \Longrightarrow (S, S_2, E_2)$ のとき，S_1 が整形されているならば S_2 は整形されている．

単一化アルゴリズム $\mathcal{DU}$ について，以下の定理が成り立つ．

定理 3.9 ($\mathcal{DU}$ の健全性・完全性). 任意の整形されている型代入 S と任意の型等式の集合 E について，もし $S(E)$ が単一化を持つならば，アルゴリズム $\mathcal{DU}$ は $S(E)$ の最も一般的な単一化 *(most general unifier)* を返す．$S(E)$ が単一化を持たないならば $\mathcal{DU}$ は失敗を報告する．

証明. まず，各変形規則は型代入環境の下で単一化の集合を保存すること，すなわち，$(S, S_1, E_1) \Longrightarrow (S, S_2, E_2)$ ならば，任意の型代入 S_0 について，S_0 が $S_1 \cup (S_1 \circ S(E_1))$ の単一化であるとき，かつそのときに限り，S_0 は $S_2 \cup (S_2 \circ S(E_2))$ の単一化であることを示す．この性質は変形規則 (u-i) および (u-ii) については明らかである．変形規則 (u-iv) についても，$t \in \mathrm{dom}(S_1 \circ S)$ であるから，$S_1 \cup (S_1 \circ S(E_1)) = S_2 \cup (S_2 \circ S(E_2))$ となり明らかである．変形規則 (u-iii) について，$t \notin \mathrm{dom}(S_1 \circ S)$ であるから，$S_1 \cup (S_1 \circ S(E_1)) = S_1 \cup (S_1 \circ$

$S(E)) \cup (S_1 \circ S(\{(t,\tau)\})) = S_1 \cup (S_1 \circ S(E)) \cup \{(t, S_1 \circ S(\tau))\}$．一方，$t \notin \mathrm{FTV}(S_1 \circ S(\tau))$ であるから，$S_3 = \{(t, S_1 \circ S(\tau))\}$ とおくと，$S_2 \cup (S_2 \circ S(E_2)) = S_3 \cup S_3(S_1) \cup (S_3 \circ S_1 \circ S(E)) \cup (S_3 \circ S_1 \circ S(\{(t,\tau)\})) = S_3 \cup S_3(S_1) \cup S_3(S_1 \circ S(E)) \cup \{(S_1 \circ S(\tau), S_1 \circ S(\tau))\}$．従って，補題 3.4 より，$S_o$ が $S_1 \circ S(E_1) \cup S_1$ の単一化であることと S_0 が $S_2 \circ S(E_2) \cup S_2$ の単一化であることは同値である．

この性質より，$\mathcal{DU}(S,E) = S'$ ならば，$(S, \emptyset, E) \stackrel{*}{\Longrightarrow} (S, S', \emptyset)$ であるから S' の最も一般的な単一化は $S(E)$ の最も一般的な単一化である．一方，補題 3.8 より S' は整形されているから，S' の最も一般的な単一化は S' 自身である．従って，$S(E)$ の最も一般的な単一化は S' である．

一方，アルゴリズムが失敗を報告する場合を考える．$\mathcal{DU}(S,E) = \mathit{failure}$ と仮定する．このときアルゴリズムの定義から $(S, \emptyset, E) \Longrightarrow (S, S_1, E_1) \not\Longrightarrow (S, S_2, E_2)$ となる $E_1 \neq \emptyset$ が存在する．ところが，変形規則の定義より，$(S, S_1', E_1) \not\Longrightarrow (S, S_2', E_2)$ ならば $S_1' \circ S(E_1) \cup S_1'$ は単一化を持たない．変形規則は単一化の集合を保存するから，$S(E)$ も単一化を持たない．

アルゴリズムの停止性については，(S, S', E) の複雑さの量を $S' \circ S(E)$ に含まれる型変数の数，E に含まれかつ $\mathrm{dom}(S' \circ S)$ に含まれない型変数の数，および E に含まれる型の大きさの総和で構成される 3 つ組で表現するとき，各変形規則が複雑さの量を必ず減少させることによって示される． □

さらに，この単一化アルゴリズムに関して，その構造から以下の性質を証明することができる．これらの補題は型推論アルゴリズムの健全性の証明に必要となる．

補題 3.10. 任意の整形されている型代入 S，任意の型等式集合 E について，$\mathcal{DU}(S,E) = S'$ ならば，$S' \circ S$ も整形されている．

補題 3.11. 任意の型代入 S，任意の型等式集合 E について，$\mathcal{DU}(S,E) = S'$ ならば，$\mathrm{dom}(S) \cap \mathrm{dom}(S') = \emptyset$ である．

補題 3.12. S を任意の整形されている型代入，E を任意の型等式集合 E とする．もし $\mathcal{DU}(S,E) = S'$ ならば，任意の型 τ について $\mathrm{FTV}(S'(\tau)) \subseteq (\mathrm{FTV}(\tau) \cup \mathrm{FTV}(S(E))) \setminus \mathrm{dom}(S')$ である．

3.4 型推論アルゴリズム $\mathcal{DW}$

型推論アルゴリズム $\mathcal{DW}$ は以下のような形の関数として定義される．

$$\mathcal{DW}(\Gamma, S, \Delta, e) = (S', \Delta', \tau)$$

$\mathcal{DW}$ は, 型環境 Γ と型環境に対する型代入環境 S，S の下で Γ から到達可能な型変数を全て含む型変数の集合 Δ, およびラムダ式 e を受けとり, 新たな型代入 S'，S' と S によって解決されていない新たな型変数の集合 Δ'，および S'，S，Γ の下でのラムダ式 e の型 τ を

$$\mathcal{DW}(\Gamma, S, \Delta, c^b) = (\emptyset, \emptyset, b)$$

$$\mathcal{DW}(\Gamma\{x : \tau\}, S, \Delta, x) = (\emptyset, \emptyset, \tau)$$

$$\begin{array}{l}
\mathcal{DW}(\Gamma\{x : \forall(t_1, \ldots, t_n).\tau\}, S, \Delta, x) = \\
\quad \text{let } \{t'_1, \ldots, t'_n\} = \mathit{newvars}(\mathrm{dom}(S) \cup \Delta, \{t_1, \ldots, t_n\}) \\
\qquad S_1 = \{(t_1, t'_1), \ldots, (t_n, t'_n)\} \\
\qquad \tau_1 = S_1 \circ S(\tau) \\
\quad \text{in } (\emptyset, \{t'_1, \ldots, t'_n\}, \tau_1)
\end{array}$$

$$\begin{array}{l}
\mathcal{DW}(\Gamma, S, \Delta, \lambda x.e) = \\
\quad \text{let } t = \mathit{newvar}(\mathrm{dom}(S) \cup \Delta) \\
\qquad (S_1, \Delta_1, \tau_1) = \mathcal{DW}(\Gamma\{x : t\}, S, \Delta \cup \{t\}, e) \\
\quad \text{in } (S_1, \Delta_1 \cup (\{t\} \setminus \mathrm{dom}(S_1)), t \to \tau_1)
\end{array}$$

$$\begin{array}{l}
\mathcal{DW}(\Gamma, S, \Delta, \mathsf{fix}\ x.e) = \\
\quad \text{let } t = \mathit{newvar}(\mathrm{dom}(S) \cup \Delta) \\
\qquad (S_1, \Delta_1, \tau_1) = \mathcal{DW}(\Gamma\{x : t\}, S, \Delta \cup \{t\}, e) \\
\qquad S_2 = \mathcal{DU}(S_1 \circ S, \{(t, \tau_1)\}) \\
\quad \text{in } (S_2 \circ S_1, (\Delta_1 \cup (\{t\} \setminus \mathrm{dom}(S_1))) \setminus \mathrm{dom}(S_2), \tau_1)
\end{array}$$

$$\begin{array}{l}
\mathcal{DW}(\Gamma, S, \Delta, e_1\ e_2) = \\
\quad \text{let } (S_1, \Delta_1, \tau_1) = \mathcal{DW}(\Gamma, S, \Delta, e_1) \\
\qquad (S_2, \Delta_2, \tau_2) = \mathcal{DW}(\Gamma, S_1 \circ S, (\Delta \setminus \mathrm{dom}(S_1)) \cup \Delta_1, e_2) \\
\qquad t = \mathit{newvar}(\mathrm{dom}(S_2 \circ S_1 \circ S) \cup (((\Delta \setminus \mathrm{dom}(S_1)) \cup \Delta_1) \setminus \mathrm{dom}(S_2)) \cup \Delta_2) \\
\qquad S_3 = \mathcal{DU}(S_2 \circ S_1 \circ S, \{(\tau_1, \tau_2 \to t)\}) \\
\quad \text{in } (S_3 \circ S_2 \circ S_1, (\Delta_1 \setminus \mathrm{dom}(S_3 \circ S_2)) \cup ((\Delta_2 \cup \{t\}) \setminus \mathrm{dom}(S_3)), t)
\end{array}$$

$$\begin{array}{l}
\mathcal{DW}(\Gamma, S, \Delta, \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2) = \\
\quad \text{let } (S_1, \Delta_1, \tau_1) = \mathcal{DW}(\Gamma, S, \Delta, e_1) \\
\qquad \sigma = \forall(((\Delta \setminus \mathrm{dom}(S_1)) \cup \Delta_1) \setminus \mathrm{FTV}(S_1(\Delta))).\tau_1 \\
\qquad (S_2, \Delta_2, \tau_2) = \mathcal{DW}(\Gamma\{x : \sigma\}, S_1 \circ S, (\Delta \setminus \mathrm{dom}(S_1)) \cup \Delta_1, e_2) \\
\quad \text{in } (S_2 \circ S_1, (\Delta_1 \setminus \mathrm{dom}(S_2)) \cup \Delta_2, \tau_2)
\end{array}$$

図 3.2: 型推論アルゴリズム

$$newvar(\Delta) = x \quad \text{such that } x \notin \Delta$$

$$\begin{aligned} newvars(\Delta, \{t_1, \ldots, t_n\}) = \text{let } & t'_1 = newvar(\Delta) \\ & t'_2 = newvar(\Delta \cup \{t'_1\}) \\ & \ldots \\ & t'_n = newvar(\Delta \cup \{t'_1, \ldots, t'_{n-1}\}) \\ \text{in } & \{t'_1, t'_2, \ldots, t'_n\} \end{aligned}$$

図 3.3: 型推論アルゴリズムの補助関数

返す．型推論アルゴリズム $\mathcal{DW}$ を図 3.2 に示す．また, $\mathcal{DW}$ の定義で用いている補助関数を図 3.3 に与える．

アルゴリズム $\mathcal{DW}$ について，以下の定理が成立する．

定理 3.13 ($\mathcal{DW}$ の健全性). S を任意の整形されている型代入，Γ を任意の型環境，Δ を任意の型変数の集合，e を任意のラムダ式とする．もし $\mathrm{FTV}(S(\Gamma)) \subseteq \Delta$，$\mathrm{dom}(S) \cap \Delta = \emptyset$, かつ $\mathcal{DW}(\Gamma, S, \Delta, e) = (S', \Delta', \tau)$ ならば，$S' \circ S(\Gamma) \rhd e : S' \circ S(\tau)$ である．

証明. 任意の整形されている型代入 S，任意の型環境 Γ，任意の型変数の集合 Δ，任意のラムダ式 e について，$\mathrm{FTV}(S(\Gamma)) \subseteq \Delta$，$\mathrm{dom}(S) \cap \Delta = \emptyset$ かつ $\mathcal{DW}(\Gamma, S, \Delta, e) = (S', \Delta', \tau)$ ならば，以下の性質が全て成り立つことを示す.

1. $S' \circ S(\Gamma) \rhd e : S' \circ S(\tau)$,
2. $\mathrm{FTV}(S' \circ S(\tau)) \subseteq (\Delta \setminus \mathrm{dom}(S')) \cup \Delta'$,
3. $\mathrm{FTV}(S'(\Delta)) \subseteq (\Delta \setminus \mathrm{dom}(S')) \cup \Delta'$,
4. $\mathrm{dom}(S') \cap \mathrm{dom}(S) = \emptyset$,
5. $\mathrm{dom}(S' \circ S) \cap \Delta' = \emptyset$,
6. $\Delta \cap \Delta' = \emptyset$,
7. $S' \circ S$ は整形されている．

証明は e の構造に関する帰納法による．

c^b の場合.

型代入の定義より $S(b) = b$ であるから，型導出規則 (const) より $S(\Gamma) \rhd c^b : S(b)$．性質 *2*〜*7* を満たすことは明らかである．

x の場合.

$\Gamma = \Gamma'\{x : \tau\}$ のとき，型代入の定義より $S(\Gamma'\{x : \tau\}) = S(\Gamma')\{x : S(\tau)\}$ であるから，型導出規則 (var) より $S(\Gamma'\{x : \tau\}) \rhd x : S(\tau)$．性質 *2*〜*7* を満たすことは明らかである．

$\Gamma = \Gamma'\{x : \forall \overline{t}.\tau\}$ のとき，アルゴリズムの定義より明らかに τ_1 は $\forall \overline{t}.S(\tau)$ の例である．一方，*newvars* の定義より $\{t'_1, \ldots, t'_n\} \cap \mathrm{dom}(S) = \emptyset$ であるから $\tau_1 = S(\tau_1)$．よって $S(\tau_1)$ は $\forall \overline{t}.S(\tau)$ の例である．従って，補題 3.5 および型導出規則 (var) より $S' \circ S(\Gamma'\{\forall \overline{t}.\tau\}) \rhd x : S' \circ S(\tau_1)$．また，前提より $\mathrm{FTV}(S(\forall \overline{t}.\tau)) \subseteq \Delta$ であるから，アルゴリズム $\mathcal{DW}$ の定義より $\mathrm{FTV}(S(\tau_1)) \subseteq \Delta \cup \{t'_1, \ldots, t'_n\}$．従って性質 *2* を満たす．性質 *3*〜*7* を満たすことは *newvars* の定義より明らかである．

$\lambda x.e$ の場合.

証明の記述を簡潔にするために $\Delta'_1 = ((\Delta \cup \{t\}) \setminus \mathrm{dom}(S_1)) \cup \Delta_1$ とおく．

newvar の定義より $t \notin \mathrm{dom}(S) \cup \Delta$ であるから $\mathrm{dom}(S) \cap (\Delta \cup \{t\}) = \emptyset$．一方，前提より $\mathrm{FTV}(S(\Gamma)) \subseteq \Delta \cup \{t\}$．以下，それぞれの性質を満たすことを示す.

性質 *1.* 帰納法の仮定より $S_1 \circ S(\Gamma\{x : t\}) \rhd e : S_1 \circ S(\tau_1)$．よって，型導出規則 (abs) より $S_1 \circ S(\Gamma) \rhd \lambda x.e : S_1 \circ S(t) \to S_1 \circ S(\tau_1)$．従って，$S_1 \circ S(\Gamma) \rhd \lambda x.e : S_1 \circ S(t \to \tau_1)$.

性質 *2.* 帰納法の仮定より $\mathrm{FTV}(S_1 \circ S(\tau_1)) \subseteq \Delta'_1$．一方，$t \notin \mathrm{dom}(S)$ であるから $\mathrm{FTV}(S_1 \circ S(t)) = \mathrm{FTV}(S_1(t))$．従って，$\mathrm{FTV}(S_1 \circ S(t \to \tau_1)) = \mathrm{FTV}(S_1(t)) \cup \mathrm{FTV}(S_1 \circ S(\tau_1)) \subseteq \Delta'_1$.

性質 *3*〜*7.* 帰納法の仮定および *newvar* の定義より明らか.

fix $x.e$ の場合.

証明の記述を簡潔にするために $\Delta'_1 = ((\Delta \cup \{t\}) \setminus \mathrm{dom}(S_1)) \cup \Delta_1$ とおく．

$\lambda x.e$ の場合と同様に，$\mathrm{dom}(S) \cap (\Delta \cup \{t\}) = \emptyset$，$\mathrm{FTV}(S(\Gamma)) \subseteq \Delta \cup \{t\}$．以下，それぞれの性質を満たすことを示す.

性質 *1.* 補題 3.5 より $S_2 \circ S_1 \circ S(\Gamma\{x : t\}) \rhd e : S_2 \circ S_1 \circ S(\tau_1)$．定理 3.9 より $S_2 \circ S_1 \circ S(t) = \tau_1$．よって，帰納法の仮定より $S_2 \circ S_1 \circ S(\Gamma\{x : \tau_1\}) \rhd e : S_1 \circ S(\tau_1)$．従って，型導出規則 (fix) より $S_2 \circ S_1 \circ S(\Gamma) \rhd \mathsf{fix}\ x.e : S_2 \circ S_1 \circ S(\tau_1)$.

性質 *2.* 帰納法の仮定より $\mathrm{FTV}(S_1 \circ S(\tau_1)) \subseteq \Delta'_1$，$\mathrm{FTV}(S_1(t)) \subseteq \Delta'_1$．*newvar* の定義より $t \notin \mathrm{dom}(S)$ であるから，$\mathrm{FTV}(S_1 \circ S(t)) = \mathrm{FTV}(S_1(t)) \subseteq \Delta'_1$．従って，補題 3.12 より $\mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_1)) \subseteq (\mathrm{FTV}(S_1 \circ S(\tau_1)) \cup \mathrm{FTV}(S_1 \circ S(t))) \setminus \mathrm{dom}(S_2) \subseteq \Delta'_1 \setminus \mathrm{dom}(S_2)$.

性質 *3.* 帰納法の仮定より $\mathrm{FTV}(S_1(\Delta \cup \{t\})) \subseteq \Delta'_1$．従って補題 3.12 より $\mathrm{FTV}(S_2 \circ S_1(\Delta)) \subseteq (\mathrm{FTV}(S_1(\Delta)) \cup \mathrm{FTV}(S_1 \circ S(t)) \cup \mathrm{FTV}(S_1 \circ S(\tau_1))) \setminus \mathrm{dom}(S_2) \subseteq \Delta'_1 \setminus \mathrm{dom}(S_2)$.

性質 *4*〜*7.* 帰納法の仮定，*newvar* の定義，$\mathcal{DU}$ に関する補題より明らか.

$e_1\ e_2$ の場合.

証明の記述を簡潔にするために $\Delta'_1 = \Delta \setminus \mathrm{dom}(S_1) \cup \Delta_1$，$\Delta'_2 = \Delta'_1 \setminus \mathrm{dom}(S_2) \cup \Delta_2$ とおく.

最初の $\mathcal{DW}$ の呼び出しに対する帰納法の仮定より $\mathrm{FTV}(S_1(\Delta)) \subseteq \Delta'_1$，$\mathrm{dom}(S_1 \circ S) \cap \Delta_1 = \emptyset$．従って $\mathrm{dom}(S_1 \circ S) \cap \Delta'_1 = \emptyset$．また，前提より $\mathrm{FTV}(S(\Gamma)) \subseteq \Delta$ であるから，補

題 3.6 より $\mathrm{FTV}(S_1 \circ S(\Gamma)) \subseteq \Delta_1'$．以下，それぞれの性質を満たすことを示す．

性質 *1.* 帰納法の仮定および補題 3.5 より $S_2 \circ S_1 \circ S(\Gamma) \rhd e_1 : S_2 \circ S_1 \circ S(\tau_1)$，$S_2 \circ S_1 \circ S(\Gamma) \rhd e_2 : S_2 \circ S_1 \circ S(\tau_2)$．また，定理 3.9 より $S_3 \circ S_2 \circ S_1 \circ S(\tau_1) = S_3 \circ S_2 \circ S_1 \circ S(\tau_2 \to t)$．従って, 型導出規則 (app) より $S_3 \circ S_2 \circ S_1 \circ S(\Gamma) \rhd e_1\ e_2 : S_3 \circ S_2 \circ S_1 \circ S(t)$．

性質 *2.* *newvar* の定義より $t \notin dom(S_2 \circ S_1 \circ S)$ であるから $\mathrm{FTV}(S_2 \circ S_1 \circ S(t)) = \{t\}$．帰納法の仮定より $\mathrm{FTV}(S_1 \circ S(\tau_1)) \subseteq \Delta_1'$，$\mathrm{FTV}(S_2(\Delta_1')) \subseteq \Delta_2'$，$\mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_2)) \subseteq \Delta_2'$．よって，補題 3.6 より $\mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_1)) \subseteq \Delta_2'$．一方，*newvar* の定義より，$t \notin dom(S_2 \circ S_1 \circ S)$．従って，補題 3.12 より $\mathrm{FTV}(S_3 \circ S_2 \circ S_1 \circ S(t))) \subseteq (t \cup \mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_1)) \cup \mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_2))) \setminus \mathrm{dom}(S_3) \subseteq (t \cup \Delta_2') \setminus \mathrm{dom}(S_3)$．

性質 *3.* 帰納法の仮定より，$\mathrm{FTV}(S_1(\Delta)) \subseteq \Delta_1'$，$\mathrm{FTV}(S_2(\Delta_1')) \subseteq \Delta_2'$．よって，補題 3.6 より $\mathrm{FTV}(S_2 \circ S_1(\Delta)) \subseteq \Delta_2'$．従って，補題 3.12 より $\mathrm{FTV}(S_3 \circ S_2 \circ S_1(\Delta)) \subseteq (\mathrm{FTV}(S_2 \circ S_1(\Delta)) \cup \mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_1)) \cup \mathrm{FTV}(S_2 \circ S_1 \circ S(\tau_2)) \cup t) \setminus \mathrm{dom}(S_3) \subseteq (\Delta_2' \cup t) \setminus \mathrm{dom}(S_3)$．

性質 *4*〜*7.* 帰納法の仮定，*newvar* の定義，$\mathcal{DU}$ に関する補題より明らか．

$\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ の場合．

証明の記述を簡潔にするために $\Delta_1' = \Delta \setminus \mathrm{dom}(S_1) \cup \Delta_1$，$\Delta_2' = \Delta_1' \setminus \mathrm{dom}(S_2) \cup \Delta_2$ とおく．

最初の $\mathcal{DW}$ の呼び出しに対する帰納法の仮定より $\mathrm{dom}(S_1 \circ S) \cap \Delta_1' = \emptyset$．同じく帰納法の仮定より，$e_1\ e_2$ の場合と同様に $\mathrm{FTV}(S_1 \circ S(\tau_1)) \subseteq \Delta_1'$，$\mathrm{FTV}(S_1 \circ S(\Gamma)) \subseteq \Delta_1'$．一方 $\mathrm{FTV}(S_1 \circ S(\sigma)) \subseteq \mathrm{FTV}(S_1 \circ S(\tau_1))$ であるから，$\mathrm{FTV}(S_1 \circ S(\sigma)) \subseteq \Delta_1'$．従って，$\mathrm{FTV}(S_1 \circ S(\Gamma\{x : \sigma\})) \subseteq \Delta_1'$．

性質 1 を満たすことを示すために，まず，σ が型導出規則 (let) に定められている条件，すなわち，$(\Delta_1' \setminus \mathrm{FTV}(S_1(\Delta))) \cap \mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) = \emptyset$ を満たすことを示す．帰納法の仮定より $\Delta_1' \cap \Delta_2 = \emptyset$ であるから，$\mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) \subseteq \mathrm{FTV}(S_1(\Delta)) \cup \Delta_2$ を示せばよい．帰納法の仮定より $\mathrm{FTV}(S_1(\Delta)) \subseteq \Delta_1'$，$\mathrm{FTV}(S_2(\Delta_1')) \subseteq \Delta_2'$．よって，前提より $\mathrm{FTV}(S(\Gamma)) \subseteq \Delta$ であるから，補題 3.6 より $\mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) \subseteq \Delta_2'$．一方，$\Delta \setminus \mathrm{dom}(S_2 \circ S_1) = \mathrm{FTV}(S_1(\Delta \setminus \mathrm{dom}(S_2 \circ S_1))) \subseteq \mathrm{FTV}(S_1(\Delta))$．従って $\mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) \subseteq \Delta_2 \subseteq \mathrm{FTV}(S_1(\Delta)) \cup (\Delta_1 \setminus \mathrm{dom}(S_2)) \cup \Delta_2$．ところで，帰納法の仮定より $\mathrm{FTV}(S(\Gamma)) \cap \Delta_1 = \emptyset$，$\mathrm{FTV}(S_1(\Delta_1)) = \Delta_1$ であるから，補題 3.6 より $\mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) \cap S_2(\Delta_1) = \emptyset$．よって，$\mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) \cap (\Delta_1 \setminus \mathrm{dom}(S_2)) = \emptyset$．従って，$\mathrm{FTV}(S_2 \circ S_1 \circ S(\Gamma)) \subseteq \mathrm{FTV}(S_1(\Delta)) \cup \Delta_2$．ゆえに，$\sigma$ は型導出規則 (let) の条件を満たす．

帰納法の仮定および補題 3.5 より $S_2 \circ S_1 \circ S(\Gamma) \rhd e_1 : S_2 \circ S_1 \circ S(\tau_1)$，$S_2 \circ S_1 \circ S(\Gamma\{x : \sigma\}) \rhd e_2 : S_2 \circ S_1 \circ S(\tau_2)$．従って, 型導出規則 (app) より $S_2 \circ S_1 \circ S(\Gamma) \rhd \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : S_2 \circ S_1 \circ S(t)$．

性質 2〜7 を満たすことは帰納法の仮定より明らかである． □

3.5 $\mathcal{DW}$ の定性的評価

1.2 節で指摘した通り，let 式を含む型推論問題の複雑さは DEXPTIME 完全であることがすでに示されており，従来のアルゴリズムよりアルゴリズム論的に効率的な型推論アルゴリズムは構成することはできない．本章で構築したアルゴリズム $\mathcal{DW}$ に対しても，Mairson の研究結果 [8] などを参考に，指数関数的な時間を要する例を容易に構築することができる．例えば図 3.4 に示すような式の型は式自体の大きさに対して非常に大きい．このような式の型を推論には，どのようなアルゴリズムを用いても指数関数的な時間が必要である．

$$
\begin{array}{l}
\mathsf{let}\ x_1 = \lambda y.\lambda z.zyy \\
\ \mathsf{in}\ \mathsf{let}\ x_2 = \lambda y.x_1(x_1(y)) \\
\quad \mathsf{in}\ \ldots \\
\qquad \mathsf{in}\ \mathsf{let}\ x_n = \lambda y.x_{n-1}(x_{n-1}(y)) \\
\qquad\quad \mathsf{in}\ x_n(\lambda z.z)
\end{array}
$$

図 3.4: 式の型が式自体よりも指数的に大きくなる例 ([8] より引用)

しかし，実用的な観点からは，アルゴリズム $\mathcal{DW}$ は型代入を遅延することで大きな型や型環境を頻繁に操作することを避けているため，多くの場合，従来の型推論アルゴリズムと比較して実用上はより高速であると期待できる．例えば，図 3.5 に示すコード例を考える[1]．従来のアルゴリズムでは，ネストした `fn` 式それぞれの型を推論するたびに，`fn` 式によって拡張された型環境に対して型代入を適用するため，およそ n^2 に比例する量に対して型代入の適用が行われると見込まれる．一方，本稿で提案するアルゴリズムでは，型環境に対する型代入の適用を行わないため，型代入の適用対象は n に比例する量に留まり，従来のアルゴリズムと比較して格段の高速化が見込める．

```
fn x => (x 1,
  fn x2 => (x2 1,
    fn x3 => (x3 1,
      ...
        fn x_n => x_n 1
        ... )))))
```

図 3.5: 冗長な型代入の適用を引き起こす例

しかしながら，アルゴリズム $\mathcal{DW}$ の実用上の優位性の議論は，多くの典型的なプログラムに対する型推論に要する時間の測定や，測定結果を元にした従来の型推論アルゴリズ

[1] この例では，説明を簡単にするために組を使っているが，組の使用は本質的ではなく，同様の例を組を使わずに構築可能である．

ムとの比較に基づく分析が必要である．さらに，現実の実用コンパイラにおいては，型推論アルゴリズムは明示的な型情報の構築やコンパイラが管理する種々の環境との複雑な相互作用を行っているため，比較評価のためのデータは，Standard ML など現実の言語に対して，実用コンパイラによるコンパイル過程において取得するのが望ましい．

第4章　実装と評価

本研究の成果として，前章で提案した型推論アルゴリズム $\mathcal{DW}$ を用いて，SML♯コンパイラ上に Standard ML の Core Syntax 相当の型推論機構を実装した．本章では，その実装の概要を述べる．また，この実装を使用してアルゴリズムの定量的な評価を行う．

4.1　実装の概要

型推論アルゴリズム $\mathcal{DW}$ の実用上の優位性や実際のコンパイラでの実現可能性を示すために，Standard ML の Core Syntex 相当の式や型を処理できるよう $\mathcal{DW}$ を拡張し，SML♯コンパイラ [16] 上に実装した．

SML♯は，大堀が提案し開発を進めている Standard ML の拡張言語である．SML♯は Standard ML の全ての構文・機能に加え，最新の研究成果に基づく数多くの拡張を含んでいる．例えば，SML♯の構文や型システムには，Standard ML の型システムを元に，ランク 1 多相性 [14] や多相レコード計算 [12] を実現するための拡張が施されている．本研究では，Standard ML の機能のうち Core Syntax に相当する部分と，SML♯による拡張のうち多相レコード計算をアルゴリズム $\mathcal{DW}$ を用いて実装した．従って，SML♯上に実装を行ったとは言え，本研究の実装が SML♯の全ての機能を有しているわけではない．Standard ML の Module Syntax に該当する機能や SML♯による拡張に対する実装は今後の課題である．

SML♯コンパイラは，ある中間言語から他の中間言語への変換を行う複数のモジュールで構成される．ソースプログラムは，構文解析を始めとする数々のコンパイルフェーズを経て種々の中間言語に変換され，最終的には目的コード言語に至る．コンパイルフェーズのうち，ソース言語に基づいた変換を行うフェーズをフロントエンド，目的コード言語に基いた変換を行うフェーズをバックエンドと言う．型推論は，フロントエンドのコンパイルフェーズのひとつである．SML♯のコンパイラフロントエンドを構成するモジュールと，それらが入力または出力とする中間言語を図 4.1 に示す．

本研究では，SML♯コンパイラのオリジナルの型推論モジュールとインターフェースを一致させ，既存の型推論モジュールを置き換える形で型推論アルゴリズム $\mathcal{DW}$ を実装した．さらに，宣言的な実装を実現するために型の定義を見直し，型を表すデータ型を再構築した．それに伴い，型推論モジュール以外のモジュールに対しても若干の変更を加えている．

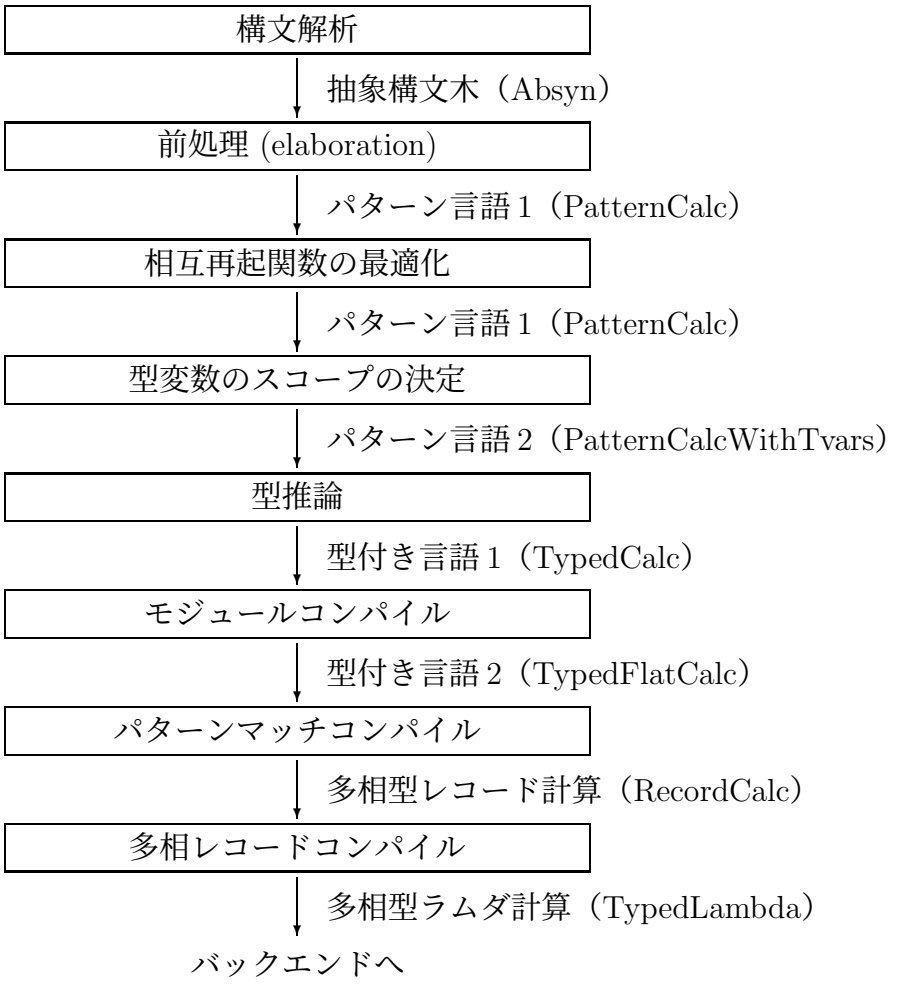

図 4.1: SML♯コンパイラフロントエンドの処理の流れ

4.1.1 型を表わすデータ型

本研究が実装の対象としたSML♯コンパイラの実装は，他の多くの関数型言語のコンパイラと同様に，型代入を破壊的な更新によって実現している．そのため，型を表現するデータ型の定義に破壊的な更新が可能な型（`ref` 型）が多く用いられている．しかし，型推論アルゴリズム $\mathcal{DW}$ は宣言的な実装が可能であるので，型が破壊的に更新可能である必要はない．従って，型を表すデータ型の定義において `ref` 型の使用が最小限となるよう，型を表すデータ型を再構築した．

この変更に追従するため，コンパイラの実装全体に渡り存在している型項を扱う関数の多くに変更を加えた．

4.1.2 型推論モジュール

型推論モジュールの実装の大部分は，Standard ML の Core Syntax や多相レコード計算を対象とするよう $\mathcal{DW}$ を自然に拡張したアルゴリズムを率直に実装したコードで構成されている．以下では，$\mathcal{DW}$ を実際の言語上に実装するにあたり，単純な拡張では不十分であった点について列挙する．

多相レコード型型推論と Δ　多相レコード型型推論 [12] には，まだ型代入によって解決されていない型変数のカインドを保存するカインド環境 $\mathcal{K}$ が必要となる．ここで $\mathcal{K}$ は，形式的には型変数からカインドへの関数である．一方，$\mathcal{DW}$ は，型変数に関する情報として，型環境から到達可能な型変数の集合 Δ を必要とする．ところが，それぞれのアルゴリズムの定義から，$\mathcal{K}$ の定義域は常に Δ と一致する．従って，$\mathcal{DW}$ を多相レコードとともに実装する場合は，Δ のかわりに $\mathcal{K}$ を使用することができるため，Δ を独立して計算したり受け渡したりする必要はない．

型代入の合成と Δ の計算　本研究で提案する型推論アルゴリズム $\mathcal{DW}$ は，頻繁に型代入の合成と Δ の計算を行っている．そのため，これらの計算方法が効率的であるかどうかが，アルゴリズム全体の効率に大きく影響する．

本研究における $\mathcal{DW}$ の実装では，型代入は型変数の ID から型へのバイナリマップで表現し，型代入の合成は，バイナリマップを線形リストで並べることで表現した．このような形式では，合成された型代入の適用時にリストの長さだけ型代入の適用を繰り返す必要がある一方，合成された型代入を表すバイナリマップを計算する必要がなくなり，全体としては型代入の適用の回数を減らすことができると思われる．また，複数の型代入を合成した型代入を表すバイナリマップを生成する必要がある場合は，補題 3.2 を利用して効率よく型代入の合成を計算できるよう工夫した．

型代入環境の適用　本研究で提案する型推論アルゴリズム $\mathcal{DW}$ は，型代入環境 S の下での型判定を返す．従って，型推論以降のフェーズで型情報が必要ならば，型推論の結果得られた型代入を何らかの形で保存し型推論以降のモジュールに渡すか，処理が次のモジュールに移る前に型代入を環境と型付き言語に適用する必要がある．前者の方針では，型環境への型代入の適用が不要となる一方，型を参照する際に毎回型代入を適用しなければならず，多くのフェーズで型主導コンパイルを行っている SML$\sharp$にとってはコードの簡潔性の点で不利である．そこで，本研究における型推論モジュールの実装では，後者の方針を採用した．この方針では型環境や型付き言語への型代入の適用が必要となるものの，そのような適用が行われるのは高々一回に過ぎず，型環境への型代入の適用を繰り返す従来のアルゴリズムに対する優位性は変わりない．また，型推論以降の型主導コンパイルフェーズでほとんど全ての型項が参照されるため，型代入の適用の更なる遅延がコンパイル処理全体の時間を短縮することには繋がらないと考えられる．

オーバーロードされた値の型　Standard ML では，定数と一部の演算子にオーバーロードを認めている．SML$^\sharp$では，これを実現するためにオーバーロードされた値の型を型変数としておき，値がオーバーロードされていることを型変数のカインドで表現している．しかし，Standard ML の定義 [10] によると，全てのオーバーロードされた値の型は型推論が終了した時点である特定の型に固定されていなければならないため，オーバーロードされていることを表す型変数を多相型の束縛変数に含めることはできない．従って，多相型 let 構文の型推論において多相型を生成するとき，$\mathcal{DW}$ の定義通りに求めた束縛変数の集合から，さらにオーバーロードされていることを表している型変数を除外しなければならない．

ユーザー定義の型変数　ユーザー定義の型変数は，そのスコープが始まる構文が現れた時点で，ユーザーの指定した名前に対応付けて型変数が生成される．このようにして生成された型変数は $S(\Gamma)$ もしくは Δ に含まれるため，$\mathcal{DW}$ では多相型 let 式の型推論において束縛の対象とはならない．しかし，Standard ML の定義 [10] によると，ユーザー定義の型変数は束縛可能でなければならない．従って，多相型 let 構文の型推論において多相型を生成するとき，束縛の対象とならない型変数の集合 $S(\Delta)$ からユーザーの定義によって生成された型変数を除外する必要がある．ユーザー定義の型変数の集合は，ユーザーの指定した名前と対応する型変数を保存している環境から得ることができる．

4.2　評価

評価は，本項の提案するアルゴリズム $\mathcal{DW}$，従来の型推論アルゴリズム $\mathcal{W}$，および現在多くの関数型言語が実装している $\mathcal{W}$ にアドホックな拡張を加えたアルゴリズムに対して行った．評価には，$\mathcal{DW}$ については本研究で SML$^\sharp$上に実装したものを，$\mathcal{W}$ については Standard ML の拡張言語である Amethyst [13] の実装を，拡張アルゴリズムについては SML$^\sharp$のオリジナルの型推論モジュールを用いた．型推論の対象とするプログラムは SML/NJ の benchmark suite [2] に対して Standard ML の Core Syntax のみを使うように変更を加えたものを使用し，それぞれの型推論アルゴリズムの実装を用いて各プログラムに対しそれぞれ 10 回ずつ型推論を行い，型推論に要した平均時間と型代入を型項に適用した回数を記録した．$\mathcal{DW}$ については，推論の結果として得られる型代入環境 S を環境と型付き言語に適用した場合と，適用しない場合の両方を計測した．評価に使用したマシンは Sun Blade 1500（CPU は UltraSPARC IIIi 1.5GHz，メモリは 1GB），使用した OS は Solaris9，使用した Standard ML の処理系は SML/NJ バージョン 110.0.7 である．評価結果を表 4.1 に示す．

SML$^\sharp$の型推論アルゴリズムでは，型変数を表すデータを破壊的に更新することで型代入の適用を実現する．そのため，実際の型代入の効果は，型変数が破壊的に更新された型項を走査するときに取り出される．この事情は，$\mathcal{DW}$ において，型推論の結果得られた型代入環境を保存しておき，型項を参照する際に型代入環境の適用を行うことに相当す

表 4.1: 型推論に要した時間 （単位: 秒）(() 内は型代入を型に適用した回数)

プログラム名	行数	SML$^\sharp$	$\mathcal{DW}$	$\mathcal{DW}$（S を適用）	$\mathcal{W}$
boyer	934	0.259	1.231 (899,160)	1.354 (942,658)	20.216 (19,160,380)
fft	222	0.033	0.126 (69,362)	0.135 (74,981)	0.618 (628,015)
knuth-bendix	592	0.147	0.456 (182,005)	0.493 (206,197)	1.781 (1,149,024)
life	154	0.054	0.178 (68,656)	0.203 (80,875)	0.515 (339,018)
mandelbrot	66	0.005	0.013 (3,710)	0.013 (4,365)	0.031 (31,748)
nucleic	3230	10.699	1.849 (1,406,689)	2.216 (1,668,696)	45.631 (5,856,353)
ratio-regions	644	0.155	0.613 (418,652)	0.654 (439,119)	4.185 (3,162,308)

ると考えられる．その一方，$\mathcal{W}$ は全ての型代入が適用された環境と型項を返す．従って，$\mathcal{DW}$ を SML$^\sharp$と比較する場合は型代入環境 S を適用していない結果を，$\mathcal{W}$ と比較する場合は型代入環境 S を適用した結果を参照するべきである．ただし，SML$^\sharp$の型推論アルゴリズムにはランク 1 多相性を実現するための拡張が含まれており，単一化や多相型の例を生成するタイミングと回数が $\mathcal{DW}$ や $\mathcal{W}$ とは大きく異なるため，単純な比較で実用上の効率の優劣を判断することはできない．しかし，SML$^\sharp$の実装も基礎のアルゴリズムには $\mathcal{W}$ を採用しており，ランク 1 多相性を推論しない場合もほぼ同様の傾向を示すと思われる．

まず，$\mathcal{DW}$ を $\mathcal{W}$ と比較すると，全ての場合において $\mathcal{DW}$ は $\mathcal{W}$ と比較して約 1/3〜1/20 の時間で型推論を完了している．また，型代入を適用する回数はほとんどの場合で 1/5 程度に抑えられている．総じて，$\mathcal{DW}$ には $\mathcal{W}$ に対して，実用上の効率の点で劇的な改善が見られた．

次に，$\mathcal{DW}$ の結果を SML$^\sharp$の結果と比較すると，ほとんどの場合において $\mathcal{DW}$ は SML$^\sharp$に比べ型推論に多くの時間を要している．しかし，その差はどのケースもほぼ一定の定数倍に収まっており，$\mathcal{DW}$ は従来のアドホックな拡張に匹敵するほどの効率を実現していると言えよう．

ところで，nucleic に関しては，$\mathcal{DW}$ は SML$^\sharp$の約 1/5 の時間で型推論を終えている．nucleic には大きな組と大きな組を扱う多相関数が含まれる．そのような大きな型の取り扱い方の違いが，型推論の要した時間に大きな差が生じた原因ではないかとは思われるものの，その詳しい原因は分からない．この差に関するより精細な調査は今後の課題である．

第5章　関連研究

5.1　従来の手法との関連性

現在，多くの関数型プログラミング言語のコンパイラが採用している型推論アルゴリズムは，Milner の型推論アルゴリズム $\mathcal{W}$ に効率化のためのアドホックな拡張を施したものである．便宜上，以下ではこの拡張アルゴリズムを $\mathcal{W}'$ と呼ぶ．$\mathcal{W}'$ は，型変数を破壊的な変更が可能な値として定義し，型代入を適用するときは，適用後の型項全体を生成し直すのではなく，型変数を破壊的に更新する．このように型変数を直接変更することによって，型環境や型項全体を走査することなく，型環境や型判定に型代入を適用するのと同じ効果を得ることができる．型を参照するときは，型を表すデータ構造を走査し，破壊的に更新された型変数から代入された型を取り出す．この戦略では非常に高速に型代入の適用を実現できるものの，アルゴリズム自体が副作用を積極的に利用する形で構成されているため，その性質を明らかにするのは難しい．

その一方で，このアルゴリズムは本研究の結果得られた型推論アルゴリズム $\mathcal{DW}$ の型代入を遅延するという戦略の一変形であるとみなすことも可能であると思われる．すなわち，$\mathcal{W}'$ は $\mathcal{DW}$ のような型代入の適用を遅延するための環境を明示的に持たない代わりにヒープを暗黙の型代入環境として利用しているとみなすことができ，また破壊的更新による型代入の適用は型代入の効果を型代入環境に保存することに，参照時に値を取り出す処理は保存した型代入を必要に応じて適用していることに対応していると考えられる．

ただし，このような対応が見出せるとは言え，$\mathcal{W}'$ と $\mathcal{DW}$ の論理的な関連性について厳密に議論することは難しい．しかしながら，本稿で提案したアルゴリズム $\mathcal{DW}$ は 2.3 節で述べた洞察にのみ基づき構築されたものであるにもかかわらず，結果として得られたアルゴリズムに従来から広く用いられているアドホックな拡張との対応が見られることは興味深い．

5.2　型推論に関する研究

Jones による単純化した Haskell の式に対する型推論アルゴリズム [4] では，型推論モナド（TI モナド）に閉じ込めた型代入に代入の効果を保存することで，型環境に型代入を適用することと同様の効果を得ている．これは，本研究が提案する型推論アルゴリズムの効率化の手法のうち，型代入の遅延に類似した最適化であるとみなせる．しかし，この論文では，このようにすることで頻繁に型代入を適用することを抑止できるとは述べている

一方，この方針の論理的な正当性には言及していない．

型推論アルゴリズムの実用上の効率化のためのアプローチとして，制約に基づく型推論アルゴリズム [11] が挙げられることがある．しかし，制約に基づく型推論アルゴリズムの効率に関する評価は，制約を解く手順に着目して行わなければならず，制約に基づくシステムについてそのような議論をすることは一般に困難である．将来，制約の解き方に着目した，制約に基づく型推論アルゴリズムの効率に関する評価が行われたならば，本研究における手法との注意深い比較が必要となるだろう．

let 多相型型推論アルゴリズムとしては，$\mathcal{W}$ の他に $\mathcal{M}$ が知られている [7]．$\mathcal{W}$ が再帰的に部分式を走査しボトムアップに型等式を構成していくのに対し，$\mathcal{M}$ は部分式が現れる文脈に従って式の型が満たすべき制約をトップダウンに構成する．しかし，制約を満たす解を単一化アルゴリズムを用いて求め，得られた型代入を型環境などに対して適用するのは $\mathcal{W}$ と同様である．従って，$\mathcal{M}$ に対しても，2.3 節で述べた洞察に基づいて，$\mathcal{W}$ と同様の効率化を施すことが可能であると思われる．

第6章　結論と今後の課題

6.1　結論

多相型型推論は有用である一方，コンパイラの複雑化とコンパイル時間の増大を招いている．今日の関数型言語のコンパイラの実装では，アドホックな拡張によって効率の改善を試みているものの，その拡張の性質は明らかではない．そこで，本研究では，高度な機能を含む最先端の関数型言語のコンパイラの高信頼かつ効率的な実装の基礎として，宣言的で実用上効率的な型推論アルゴリズム $\mathcal{DW}$ と，それを構築するための単一化アルゴリズム $\mathcal{DU}$ を提案した．アルゴリズム $\mathcal{DW}$ では，従来の型推論アルゴリズム $\mathcal{W}$ を元に，項や環境への型代入の適用を遅延させるなどして大きな項や項の集合を操作する回数を減らすことで，アルゴリズムの実用上の性能の改善を試みた．その結果，アルゴリズム論的には DEXPTIME 完全であるものの，現実のプログラムに対する実用上の性能の向上が期待できた．さらに，提案したアルゴリズムの健全性をそれぞれ示した．

また，型推論アルゴリズム $\mathcal{DW}$ を SML$\sharp$コンパイラ上に実装し，$\mathcal{DW}$ が現実のプログラミング言語のコンパイラに適用可能であることを示すとともに，その性能を定量的に評価した．評価の結果，現実のプログラミング言語において，$\mathcal{DW}$ は従来のアルゴリズム $\mathcal{W}$ と比べて非常に高速であり，また従来のアドホックな手法と比較しても同程度の効率を有しているという結果を得た．

6.2　今後の課題

アルゴリズム自身の改良点として，例えば以下のような可能性が考えられる．$\mathcal{DW}$ では，部分式の型推論の結果得られた差分 S'，Δ' から，新しい S，Δ を構築する処理を頻繁に行っている．しかし，実際に型代入や型変数を生成するのは単一化を行う場合など，アルゴリズムの一部に過ぎない．そこで，$\mathcal{DW}$ を S，Δ の差分ではなく新しい S，Δ 全体を返すように変形し，S，Δ に対する変更は実際に型代入や型変数を生成している部分に集約すれば，より一層の効率の向上が得られる可能性がある．

また，アルゴリズム $\mathcal{DW}$ で扱う型代入はすべて整形されているものであるため，$\mathcal{DW}$ では型変数への型代入の適用は常に 1 回で停止する．しかし一方で，受け取る型代入が整形されていることを必要条件とせず，またアルゴリズム内部で型代入が整形されていることを保存しないようなアルゴリズム $\mathcal{DW}$ の変形も考えられる．これは，明示的代入操作をもつラムダ計算（例えば [1] などを参照）などで行われている代入に関する簡約規則に

相当する操作と見なすことができる．整形されていない型代入の型項への適用には型代入の推移的な適用が必要である反面，このようにアルゴリズムを変形することで vacuous な型変数 (vacuous type variable)[12] に対する不要な型代入の適用が抑止できることが期待でき，全体として効率が向上する可能性がある．

一方，単一化アルゴリズム $\mathcal{DU}$ では，変形規則が適用される順序は非決定的である．そのため，単一化アルゴリズム $\mathcal{DU}$ の効率性について厳密に議論することはできない．単一化を求めるにあたり，変形規則の適用順序を制御することができれば，アルゴリズムの振る舞いをより明確にすることができるとともに，アルゴリズムの効率を向上させることも可能かもしれない．

謝辞

本研究を進めるにあたり，熱心に御指導を賜りました大堀淳教授に深く感謝致します．本稿は大堀淳教授による御指導なしには決して完成することはなく，また本研究に限らず博士前期過程におけるあらゆる面で多大なご助言，ご協力を賜りました．また，実装にあたりご協力を賜りました，IML プロジェクトの皆様に感謝致します．最後に，随所で熱心な議論とご指導を頂きました友人諸氏，ならびに計算機言語学講座の皆様に厚く御礼申し上げます．

参考文献

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J Levy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] Benchmark suite for Standard ML. `ftp://ftp.research.bell-labs.com/dist/smlnj/benchmarks/`.

[3] J. Gallier and W. Snyder. Complete sets of transformations for general E-unification. *Theoretical Computer Science*, 67(2):203–260, 1989.

[4] M. P. Jones. Typing Haskell in Haskell. In *Proceedings of the 1999 Haskell Workshop*, Paris, France, October 1999. Published in Technical Report UU-CS-1999-28.

[5] P. Kanellakis, H.G. Mairson, and J. Mitchell. Unification and ML type reconstruction. In J-L Lassez and G. Plotkin, editors, *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1990.

[6] A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *Journal of the ACM*, 41(2):368–398, 1994.

[7] O. Lee and K. Yi. Proofs about a folklore let-polymorphic type inference algorithm. *ACM Transactions on Programming Languages and Systems*, 20(4):707–723, 1998.

[8] H. G. Mairson. Deciding ML typability is complete for deterministic exponential time. In *POPL*, pages 382–401, 1990.

[9] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[10] R. Milner, R. Tofte, M. Harper, and D. MacQueen. *The Definition of Standard ML*. The MIT Press, revised edition, 1997.

[11] M. Odersky, M. Sulzmann, and M. Wehr. Type inference with constrainted types. *Theory and Practice of Object Systems*, 5(1):35–55, 1999.

[12] A. Ohori. A polymorphic record calculus and its compilation. *ACM Transactions on Programming Languages and Systems*, 17(6):844–895, 1995. A preliminary summary

appeared at ACM POPL, 1992 under the title "A compilation method for ML-style polymorphic record calculi".

[13] A. Ohori and K. Yamatodani. An interoperable calculus for external object access. In *ICFP '02: Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, pages 60–71, New York, NY, USA, 2002. ACM Press.

[14] A. Ohori and N. Yoshida. Type inference with rank 1 polymorphism for type-directed compilation of ML. In *Proc. ACM International Conference on Functional Programming*, pages 160–171, 1999.

[15] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of ACM*, 12:23–41, March 1965.

[16] SML$^\sharp$ Compiler. `http://www.pllab.riec.tohoku.ac.jp/smlsharp/`.