プログラミング言語Standard ML入門 改訂版
大堀 淳著
https://www.kyoritsu-pub.co.jp/bookdetail/9784320124806

目　　次

プログラミング言語Standard ML入門 改訂版
大堀 淳著
https://www.kyoritsu-pub.co.jp/bookdetail/9784320124806

プログラミング言語Standard ML入門 改訂版
大堀 淳著
https://www.kyoritsu-pub.co.jp/bookdetail/9784320124806

ML の特徴と本書の構成

ML は手続き型のプログラミング言語にはないいくつかの特徴を持っている．それらの特徴は ML の歴史的背景に由来するものが多く，それらが導入された意図を歴史的背景とともに理解することは，ML を習得する上でも有益と思われる．そこで，ML の解説に入る前に，本章にて，その特徴を歴史的背景とともに概説する．その後，本書の構成と学び方および参考書について記しておく．

ML の歴史的背景とその特徴

ML は，Edinburgh 大学で 1970 年代の後半に，証明導出システム Edinburgh LCF のための記述言語として開発された．LCF システムの記述の対象は，定理や公理，推論規則などであり，ML は，それらを計算機システムの中で表現，操作するための言語，すなわちそれら数学的対象が属する形式言語のメタ言語 (metalanguage) であった．その名前のみならず，ML の特徴の多くは LCF のメタ言語としての要求に由来している．

LCF システムのプログラムは，公理や推論規則を合成し効率よく証明を導出するための証明戦略などで構成されている．推論規則は命題から命題を導出する関数と考えられるため，それらを操作する証明戦略は，関数を合成し新たな関数を生成するような高階の関数として表現するのが最も自然である．この理由から，LCF ML は，高階の関数を自由にデータとして扱うことができる関数型言語として設計された．

LCF システムに限らず，種々の証明導出システムや推論システムは，同様の理由から，Lisp や Scheme などの関数型言語で書かれていることが多い．このようなほかの関数型言語と比較した LCF ML の最大の特徴は，プログラムの型を自動

的に推論する静的型システムを装備していたことである．

証明戦略などは，通常，種々の場合に適用可能な一般性のある規則であるため，それらの表現も，その一般性を反映した汎用性ある関数であることが望ましい．一方，大規模なシステムの信頼性の向上のためには，複雑なプログラムに潜在する誤りの多くをコンパイル段階で検出することができる静的型チェックも強く望まれる．しかし，汎用性ある関数の表現と静的な型チェックというこの 2 つの要求は互いに相反する性質のものであり，従来，汎用な関数の定義が可能な Lisp などの言語では，静的な型チェックは不可能であった．

LCF ML の型システムの設計者であった Robin Milner は，複雑なプログラムに潜む不整合をコンパイル段階で自動検出することを目的として，高階の関数を含むプログラムの最も一般的な型を自動的に推論する型推論の理論とその実用的アルゴリズムを構築した．この理論は，型宣言を含まないプログラムから，そのプログラムの持つ汎用性を正確に表現する「最も一般的な多相型」を自動的に推論することができる画期的なものであった．LCF ML は，この型推論理論を基礎に設計された静的型システムによって，型付き言語の安全性と信頼性を損なうことなく Lisp などの型なし言語（動的に型付けられた言語）の柔軟性を実現し，LCF システムのプログラミングの信頼性と柔軟性の向上に大きく貢献した．

LCF ML の成功はプログラミング言語の研究者の注目を集めた．特にその型システムは，証明導出システムのメタ言語としてばかりでなく，汎用のプログラミング言語にとっても有用性が高いと認識され，それ以降，LCF ML を拡張し汎用プログラミング言語を設計する努力が行われた．それらの中で今日の ML に採用されている重要な成果は，パターンマッチングとモジュールシステムである．

パターンマッチングは，1970 年代に Edinburgh 大学で開発されたプログラミング言語 HOPE で提案された．これは，処理したいデータの形をパターンを用いて記述することによって，複雑なデータの操作の記述を容易にする機構である．Milner は，このパターンマッチング機構を LCF ML の多相型システムと統合し，現在の ML の骨格となるプログラミング言語 The Standard ML Core Language を定義した．一方 David MacQueen は，HOPE 言語にあったモジュールの考えを一般化し，モジュールのための型システムを構築し，それを基に ML のためのモジュールシステムを設計した．

これら 2 つの定義はその後改定され，Robert Harper によるストリームに基づく入出力処理の記述とともに，当時 LCF や ML および HOPE 言語などに興味を持つ研究者たちの間のニュースレターであった *Polymorphism* にまとめて発表された．この 3 つのレポートが今日の Standard ML の原型である．それ以降，この提案に対して注意深い検討と改定およびその整合性の理論的な検証がなされ，1990

年に ML 言語を定義した *The Definition of Standard ML* が出版された．この定義にはさらなる改良が加えられ，1997 年に第 2 版が出版されている．これが現在の Standard ML の仕様である．

ほかの言語の定義と違い，この Standard ML の定義は型理論の枠組みで厳密に記述されており，この定義に基づき，多くの望ましい性質が数学的に証明されている．たとえば，ML で書かれたプログラムは，実行時に未定義の操作を実行してシステムを停止させるようなことは起こりえない．この性質は，ML で書かれたプログラムに高い信頼性を与えている．ML の意味論の形式的な定義はまた，プログラミング言語の新しい機能を考える上での理論的基礎ともなっている．現在 ML を基礎にして，並行計算，オブジェクト指向計算，データベース操作などの機能をプログラミング言語に統合する研究が行われている．

Standard ML の仕様決定および仕様の理論的検証と並行して，ML 処理系の実装の努力が行われた．LCF ML 以後の，独立したプログラミング言語としての最初の ML コンパイラは，おそらく Luca Cardelli のシステムであろう．この実装の一部を基礎とし，Edinburgh 大学のグループによって，上記の 3 つのレポートに定義された仕様をほぼ満たす Standard ML コンパイラが作成され，大学や研究機関などで使用された．その後 AT&T の MacQueen と Princeton 大学の Andrew Appel が中心となって開発した Standard ML of New Jersey コンパイラは，Standard ML の仕様を満たす効率よいコードを生成するシステムであり，実用言語としての Standard ML の普及に大きく貢献した．またフランスの INRIA 研究所においても，Xavier Leroy によって，ML 系言語の一つである Objective Caml（OCaml）が開発されている．残念ながら OCaml の文法は Standard ML とは互換性がないが，Leroy によるコンパイラは，効率よいコードを生成する質の高い ML 系言語の処理系である．わが国では，北陸先端科学技術大学院大学および東北大学において，大堀と上野が，実用的なシステム開発に適した SML# コンパイラを開発した．SML# は，Standard ML の言語仕様の不十分さを補うレコード多相性などを導入し，さらに，伝統的な ML 言語では不得意な C 言語やデータベースとの連携，マルチコア上の並列スレッドなどの実用的な機能を装備した Standard ML の拡張言語である．それ以外にも現在複数の処理系が作成されている．

本書の構成

通常 ML 言語といえば，言語仕様とそれを実現するコンパイラをさす．しかし，ML で本格的システム開発を行うためには，種々のデータ構造の操作，入出力をはじめとする種々の資源の操作，種々の標準のツールなどを提供するライブラリの存

在が必須である．Standard ML では，豊富なライブラリ群が Standard ML 基本ライブラリ (Standard ML Basis Library) として提案され提供されている．Standard ML にて本格的なプログラムを書いていくためには，その言語仕様のみならず，基本ライブラリの習熟も重要である．そこで本書は，Standard ML の言語を解説する第 I 部と基本ライブラリを解説する第 II 部の 2 部構成とする．

第 I 部では，対話型システムの使い方から関数や種々のデータ構造，型システムなどの Standard ML 言語の特徴と機能を，SML# コンパイラを使用して，段階的に解説する．それらの理解のまとめとして，第 11 章で，Standard ML 言語の文法定義を含める．第 II 部では，標準ライブラリの中で，すべての Standard ML 言語処理系が提供すべき必須モジュールの概要および，主要な型と関数の使用方法を例を用いて説明する．

ML でのシステム開発には，Standard ML 言語や基本ライブラリの理解に加えて，各コンパイラが提供する実行コマンドの作成方法やシステムライブラリとのリンク方法などを理解する必要がある．そこで，第 18 章 付録で，SML# の入手方法を説明したあと，コマンドの作成や分割コンパイル，外部ライブラリとのリンクなどの SML# の基本機能を説明する．この付録および，Standard ML の言語仕様の不十分さを補う SML# の拡張機能である 3.4 節の後半のオーバーロード多相性，5.4 節のレコード多相性，8.6 節のランク 1 多相性の各解説以外は，ほかの Standard ML のコンパイラにも共通である．

解説の一部として，テキストの文中に適宜練習問題を含めた．これらの問題を含め，最初からページの順に読み進んでいくことが可能なテキストとなるよう心掛けた．SML# コンパイラをインストールしたパソコンなどを活用し，これらの練習問題を解きながら，はじめから最終章まで順に一気に読み通されることを勧める．ある程度まとまった時間があれば，2〜3 週間程度の短時間で ML プログラミングを身につけることができるはずである．

ML に関する参考書

最後に，さらに学習したい読者のために，ML に関する文献を紹介しておく．

本書で ML 言語に興味を持った者が，ML での本格的なプログラミングを学ぶためには，改訂版の序文で言及した以下の姉妹書が参考になる．

- 大堀 淳・上野雄大 (2021). SML# で始める実践 ML プログラミング．共立出版．サポート Web ページ：`https://smlsharp.github.io/ja/textbooks/`

この教科書は，本書ではカバーされていない ML 流のプログラミングの考え方や

システム設計技法，さらに，データベースとの連携や並列プログラミングなどの SML# の機能を駆使した実用的なシステム構築技術を，例を用いて解説している．

ML プログラミングについては，欧米で多数の教科書が出版されている．ここでは，本書や上記姉妹書以外にさらに ML プログラミングを学んでみようと思う者のために，以下のものを挙げておく．

- Paulson, L. C. (1997). *ML for the Working Programmer* (*2nd edition*). Cambridge University Press.

Paulson は定理自動証明システムの研究者であるとともに，ML の発展にも貢献しており，大規模システムを ML で実装した経験を持つ．この本は，ML をよく知る研究者によって書かれた良書である．

本章で紹介した Standard ML の厳密な定義は以下の書籍として出版されている．

- Milner, R., Tofte, M., Harper, R. and MacQueen, D. (1997). *The Definition of Standard ML* (*revised*). The MIT Press.

構文構造，型システム，プログラムの操作的意味（動作) が，型理論の枠組みで厳密に定義されている．ただし，この本を読みこなすには型付きラムダ計算に関する理解を必要とする．この本は，主に，ML コンパイラの実装者や ML の研究者のための参考書である．

Standard ML 基本ライブラリの仕様は以下の書籍として出版されている．

- Gansner, E. R. and Reppy, J. H. (2004). *The Standard ML Basis Library*. Cambridge University Press.

ライブラリの仕様のみならず，その使用法に関する解説もなされている．

言語ライブラリの仕様以外にも，ML 言語やその型システムに関する文献が多数出版されているが，そのほとんどはプログラミング言語の研究者向けのものであり，初めて ML を学ぼうとする多くの者には専門性が高いと思われるので，ここでは，ML の特徴，歴史および研究動向を解説した以下の文献を紹介しておく．

- 大堀 淳 (1994).「ML–多相型システムをもつ関数型言語–」．情報処理，**35**(3).

この文献には多数の参考文献が含まれているので，ML を詳しく学びたいと思う者にとっての手がかりとなるであろう．