SML#：最先端の機能と高い実用性を実現する次世代多相型プログラミング言語

大堀　淳 ＊1

＊1 東北大学電気通信研究所

● A ● B ● S ● T ● R ● A ● C ● T ●

SML# は，文部科学省委託研究 e-Society 基盤ソフトウェアの総合開発の課題「プログラム自動解析に基づく高信頼ソフトウェアシステムの構築技術」の主要な実現目標の 1 つとして，東北大学電気通信研究所が（有）算譜工房と共同で開発してきた ML 系次世代高信頼プログラミング言語である．レコード多相性，ランク 1 多相性，データベースの多相型理論，相互運用性の型理論，型主導コンパイル理論などの我々の基礎研究成果に基づく種々の機能を含む最先端の ML 系言語であるばかりでなく，C 言語や Java，データベースとのシームレスな連携，多バイト文字処理等のライブラリ，自動プログラミングツールなどをサポートする実用性の高い言語を目指している．本稿では，SML# の開発の背景と意図を概観した後，SML# の特徴をプログラム例などを用いて紹介する．

次世代高信頼言語の必要性

インターネットの普及に伴い，ネットワークで有機的に結合され協調して動作する大規模で高機能なソフトウェアを短時間で効率よく構築する必要性が高まっている．そのためには，外部資源を動的に利用し協調して動作する複雑で高機能なソフトウェアを効率よく記述可能な，高水準高信頼プログラミング言語の実現が急務である．しかし現状では，このような要求に十分に応え得るプログラミング言語は実現されておらず，その結果，アプリケーションの多くは，型チェックが十分ではない言語でプログラムされている．それらの言語で書かれたプログラムは，その動作の正しさの静的解析と検証が不十分であるため，完成した後もシステムに多くのプログラミングエラーが潜在する脆弱性を持つ．

信頼性確保のためには，膨大な時間をかけたプログラムのテストとデバッグ，さらにシステムの完成後も除去しきれなかったプログラムエラーに起因する障害の修正を含む保守が必要であるが，現在需要が急速に拡大しつつある Web アプリケーションのような短納期が求められるシステムにとっては現実的ではない．プログラムに潜在するエラーをコンパイル時に自動的に検出し取り除くための基盤技術の確立は，ソフトウェア開発の生産性の向上とテストや保守コストの低減に寄与し，IT 産業，さらには社会の生産性の向上に大きく寄与すると考えられる．

プログラミング言語 ML は，型理論を基礎とするプログラムの静的解析技術により，プログラムの型の整合性の解析と検証を行い，多くのエラーをコンパイル時に自動的に検出することができる高信頼言語である．さらに ML は，複雑なソフトウェアの生産性の向上に大きく寄与する以下のような特徴を持つ．

- 多相型システム
  多相型を含む強力な静的型システムを持ち，Lisp などの型なし言語と比肩し得る柔軟で汎用性あるプログラムを書くことができる．
- データ型の定義と利用
  関数を含む種々のデータ型をユーザが自由に定義する機構と，それらデータ型を利用するパターンマッチング機構が提供されている．さらに，自動ごみ集め機構（Garbage Collection）により，データ領域の確保と解放はシステムによって自動的に行われる．
- 高信頼モジュールシステム
  高機能なモジュールシステムが，言語の多相型システムと一体化して実現されている．これによって，モジュール間のインタフェースの不整合はすべてコンパイル時に自動検出される．

ML 系言語のこれらの優れた機能は，複雑なソフトウェア開発の信頼性と生産性を飛躍的に高める可能性を持つ．しかしながら，現在，ML 系言語が実用アプリケーションの開発用言語として広く普及しているとは言いがたい．その原因の 1 つは，Standard ML of New Jersey や Objective Caml などの現在の代表的な ML 系言語処理系が含む制約や制限にあると考えられる．実用プログラム開発にとって特に問題と考えられる弱点は以下の 2 つである．

- 相互運用性の欠如
  ML に限らず，関数型言語の通常の機能（対話型プログラミング，高階関数等，自動ごみ集め）を実現して

いる現在の処理系は，そのデータ表現に関して，C などの既存言語との相互運用性を持っていない．たとえば Objective Caml や Standard ML of New Jersey では，最も基本的なデータである整数型は，計算機による自然な表現（通常のアーキテクチャでは 2 の補数で表現された 32 ビットの符号付き整数）ではなく，言語処理系の実装上の都合による 1 ビットのタグを含む 31 ビット表現であり，また，浮動小数点データは，ヒープに割り当てられたポインタで表現されレコード（構造体）などに直接格納することはできない．いかに最先端の機能を有していようとも，整数や浮動小数点データなどの基本データを自然に表現できない言語を，実用プログラム開発に安心して採用できるか疑問である．

- レコードの扱いの制限

  ML 系高信頼言語の最大の特徴は，その多相型型推論機構にある．ML では，プログラマに複雑な型宣言を要求することなく，プログラムの汎用性を正確に表した多相型が推論され，柔軟で高機能なプログラミングが可能となっている．しかしながら，現在の ML 系言語では，この機能はレコード構造には対応しておらず，レコード処理に関する多相型プログラムを書くことができない．レコード構造は，データベースのタプルなどの表現に使用されるなど，実用プログラムにとって最も重要なデータ構造の 1 つである．レコード操作に対する多相型型推論機能の欠如は，ML 系言語の実用化における大きな制約である．

次世代高信頼ソフトウェア生産の基盤の確立の 1 つの鍵は，これら弱点を克服し，実用性の高い ML 系次世代高信頼プログラミング言語を開発することである．この社会的要求に応えることを目的とし，東北大学電気通信研究所では，ML 系高信頼言語によるシステム開発能力を有する数少ない企業である（有）算譜工房と共同で，次世代 ML 系高信頼言語 SML# の開発を進めている．東北大学の持つ型理論や型主導コンパイル等の基礎理論と算譜工房の持つソフトウェア開発のノウハウを活かした開発体制をとり，SML# コンパイラの開発のみならず，ML でのプログラム開発の生産性の向上に寄与する種々のプログラミングツールの開発に成功するなどの相乗効果を挙げている．

SML# 言語の開発方針

SML# は，前述の既存の ML 系言語の弱点を克服し，さらに我々のプログラミング言語の基礎研究で得られた種々の最先端機能を実現している．本章では，SML# の設計方針および新たに実現する機能の概要を紹介する．

【 Standard ML との上位互換性の保証 】

我々の目的は，先端機能を実現する実用性の高い次世代 ML 系言語の実現である．そのような言語として SML# を設計する上で我々が重視した方針は，既存の ML 系言語と完全な上位互換性を保証することである．これによって，すでに開発されているライブラリやツールなどを継続して使用できることが保証されるため，安心して新たな言語に移行することが可能である．この性質は，ソフトウェア生産言語として選択肢の 1 つになり得るための重要な要素と考える．

現在広く使われている ML 言語には，Objective Caml と Standard ML がある．Objective Caml コンパイラは広く使用されきわめてすぐれた言語処理系であるが，その処理系が実現する言語の仕様の厳密な定義は，フランスの INRIA 研究所で開発されているコンパイラのソースコードそのもの以外存在しないと言ってよい．そのため，Objective Caml と厳密に上位互換性を保つ新たな言語を設計実装するのは，原理的に不可能である．これに対して，Standard ML は，その言語仕様の詳細が The Definition of Standard ML として定義され出版されている．この定義は，型理論の枠組みによって厳密になされている．さらに，Standard ML のためのライブラリ仕様も Standard ML Basis Library として定義され出版されている．

この点を考慮し，SML# を，Standard ML と上位互換性を持つ言語として開発する方針とした．SML# に導入されたランク 1 多相性やレコード多相性などの先端機能は，構文論的にも意味論的にも，The Definition of Standard ML で定義された言語仕様の制限や変更がなされないように，慎重に導入されている．SML# のサポートライブラリは，Standard ML Basis Library で必須と規定されたライブラリ群をすべて含むものとして提供されている．SML# コンパイラは，これら仕様に従って作成された数千に及ぶテストケースによってテストされ，その上位互換性が確認されている．

この上位互換性の確保は，高信頼基盤ソフトウェアの開発言語として広く使用されることを目指した実用言語の実現を目指す上で必須の要素と考える．

【 スクラッチからの開発 】

SML# の実装技術は，Standard ML や Objective Caml の実装技術と共通する点が多く，開発にあたっては，それら既存の処理系のコードの流用も，選択肢の 1 つとなり得る．しかし我々は，以下の点を考慮し，汎用の定型的なライブラリを除き，すべてをスクラッチから開発する方針とした．

- 既存の処理系が用いている実装技術が，相互運用性の

欠如などの現在の ML 系言語処理系の弱点の 1 つの原因になっている．

- システムの信頼性と拡張性を確保する上で，コンパイラが使用するアルゴリズムとデータ構造の詳細をすべて把握しておく必用がある．
- 先端機能を実現するコンパイラを一から開発することによって，SML# に限らず将来のコンパイラの設計や実装にとって有益な新しい技術や理論が生み出されると期待される．

この方針の下に，我々は，SML# の構文解析法規則からガーベジコレクタに至るすべてのコードを開発した．SML# 開発過程で生まれた理論の章で紹介するとおり，実際にその過程で，汎用性のある新たな技術が複数生み出されている．

【 先端機能の実現 】

SML# は，前節で紹介した Standard ML のすべての機能に加え，現在の ML 系言語の弱点を補いその可用性を大幅に向上させる実用性の高い種々の新しい機能を取り入れることを目指した．その代表的なものは以下の 3 つである．

- 多相型レコード演算
  SML# では，我々が構築したラベル付きレコード構造を操作するための型理論[3]を基礎とし，ラベル付きレコード構造を扱うプログラムに対しても多相型推論を実現する．この機能により，レコードを含むデータ構造の柔軟な取り扱いが可能となり，さらに，データベースやオブジェクト指向言語とのよりシームレスな相互運用が可能となる．
- ランク 1 多相性
  ランク 1 多相性[6]とは，型変数のスコープを導入することによって，データ構造の部分も多相型を持つことを許すように拡張し，より一般的な型を推論する機構である．これによって，副作用を伴うプログラムの型の制約を大幅に緩和することができる．
- 高い相互運用性
  大部分の大規模システムは，既存のプログラム部品やライブラリを必要とするため，現在の ML 系言語の相互運用性の欠如は実用性を制限する大きな問題点である．SML# では，C，Java，およびデータベースシステムに対する高い相互運用性を実現する．
  - C との相互運用性の確保
    システム記述言語である C との相互運用性の確保は重要な課題である．SML# では，従来のようにデータの変換による C との連携ではなく，整数型や浮動小数点データのような基本データ型を含む標準的なデータ型の表現を C と同一にすることによって，C とのシームレスな相互運用性を実現する．
  - Java との相互運用性
    Java との相互運用のための型理論を基礎とし，Java との型システムレベルでの相互運用性を実現する．さらに，Java と相互運用性を持つごみ集め方式を実現し，Java との高い相互運用を実現する．
  - データベースとの相互運用性
    データベースの多相型理論[1]を基礎とし，多くの実用プログラムで必要とされるデータベースとの連携を実現する．

これらの機能は，実用上重要であるにもかかわらず，既存の言語ではまだ実現されていないものである．

SML# の新機能とプログラム例

本章では，SML# が新たに実現した機能の中から，レコード多相性と C 言語とのシームレスな連携機能を紹介したのち，SML# の機能を使って実現したプログラムを紹介する．

【 レコード多相性 】

レコードは属性名を表すラベルとその値の組の集合であり，C の構造体やデータベースのタプルに相当するデータ構造である．たとえば以下の例は X 座標と Y 座標を持つ点 point を定義し，その X フィールドを取り出している．

```
# val point = {X = 1, Y = 2};
val point = {X = 1 ,Y = 2} : {X: int,Y: int}
# #X point;
val it = 1 : int
```

このような単相型のプログラムは既存の処理系でも書くことができるが，既存の ML 系言語の処理系ではレコードを扱う多相型関数は定義できない．SML# ではこの問題を完全に解決している．SML# では，l フィールドを含むレコード e_0 の l フィールドを返す式 #l e, や l フィールドを含むレコード e_0 の l フィールドを e_1 に変更して得られるレコードを返す式 e_0 # $\{l=e_1\}$ に対して，その式の持つ正確な多相型が推論される．**図 -1** にレコードを扱う多相関数を定義する対話型セッションの例を示す．getX 関数に対して推論された型 `['a,'b#{X:'a}.'b ref -> 'a]` は，は，getX が X フィールドを含む任意の型のオブジェクト（ref 型のレコード）を受け取り，X フィールドの値を返す関数を表している．これによって，getX を種々の構造を持つオブジェクトに適用することができる．

```
# val getX = fn self => #X (!self);
val getX = fn : ['a,'b#{X:'a}.'b ref -> 'a]
# val setX = fn self => fn x =>
              self := (!self # {X = x});
val setX = fn : ['a,'b#{X:'a}.'b ref -> 'a -> unit]
# fun moveX p dx = setX p (getX p + dx);
val moveX = fn
  : ['a#{X: int}.'a ref -> int -> unit]
```

図 -1　レコードを扱う多相関数の例

```
$ wget http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES/mt19937ar.tgz
$ tar xvfz mt19937ar.tgz; gcc -shared -o mt19937ar.so mt19937ar.c
$ smlsharp
# structure MT = struct
> local val mtLib = DynamicLink.dlopen "./mt19937ar.so"
> in val init_genrand = DynamicLink.dlsym (mtLib,"init_genrand") : _import (int) -> void
>    val genrand_res53 = DynamicLink.dlsym (mtLib,"genrand_res53") : _import () -> real
> end end;
structure MT : sig val genrand_res53 : unit -> real
                   val init_genrand : int -> unit
               end
# MT.init_genrand 192346; MT.genrand_res53();
val it = 0.503821863109 : real
```

図 -2　SML# の C との相互運用機能を利用したプログラム例

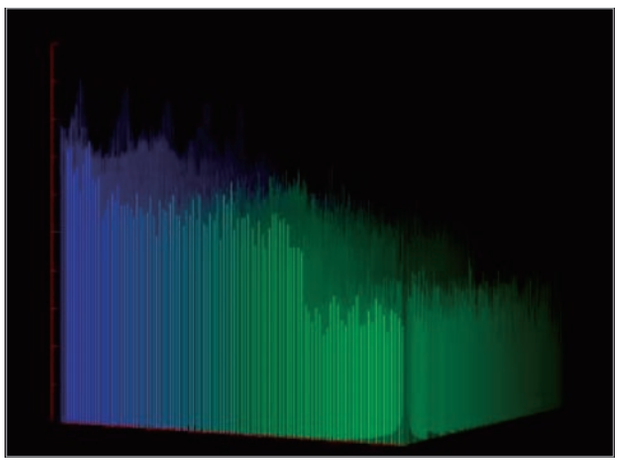

図 -3　SML# プログラムの出力例

【 C との相互運用機能 】

SML# は，型主導コンパイルの理論と実装技術により，整数や浮動小数点データに対する自然な表現を実現している．この性質による，C で実装された関数をインポートし SML# で定義されたデータ型や関数に直接適用することができる．さらに，SML# の関数表現を C の関数として呼び出せる形式に自動変換する理論と技術を実装している．この機能を使えば，コールバック関数が必用な C 言語のライブラリなども簡単に使用することができる．

SML# では，C の関数をインポートするため，C 言語のライブラリ関数 dlopen および dlsym と同等の以下の基本関数が提供されている．

```
DynamicLink.dlopen : string  -> unit ptr
DynamicLink.dlsym
 : unit ptr * string -> unit ptr
```

ユーザは，C の関数を動的リンクライブラリとして作成しさえすれば，これら機能を使ってインポートし，以下のような型宣言をすることにより，SML# から直接呼び出すことができる．

```
exp : _import (t1,...,tn) -> t
```

たとえば，松本らによる C で実装された優れた乱数生成プログラム Mersenne Twister も，**図 -2** のようにダウンロードし，SML# から直接呼び出すことができる．

【 SML# プログラム例 】

SML# が提供する C とのシームレスな相互運用性などの機能を使えば，各種ライブラリを用いて，実用的なプログラムを効率よく開発することができる．**図 -3** は，以下の処理を行う SML# プログラムの出力例である．

1. C 言語の fdopen(3) 関数と fread(3) 関数を用いて標準入力から符号付き 16 ビットステレオの生オーディオストリームを読み込む．
2. SML# プログラムでオーディオデータを浮動小数点数の配列に変換して，窓を掛ける．
3. GSL の FFT 関数に浮動小数点数の配列を渡してフーリエ変換を行う．

4. SML# プログラムでフーリエ変換の結果からパワースペクトラムの計算し，デシベルへ単位を変換する．
5. パワースペクトラムをヒストグラムとして描画するための浮動小数点計算を行い，C 言語の OpenGL および GLUT ライブラリを利用して画面に表示する．

既存の ML 系言語では，1，3，5 のステップを，コンパイラの特別なサポートや特別なライブラリなどを開発することなく行うのは困難である．

SML# 開発過程で生まれた理論

SML# の新たな機能の実現や既存の ML 系言語コンパイラの弱点の克服を目指す中で，いくつかの新しい技術や理論が生み出された．ここでは，その主なものを 3 つだけ紹介する．興味のある読者は，SML# の Web ページを参照されたい．

- 機械語コードの証明論的基礎 [4]
  機械語コードを直観主義的論理学の証明記述と見なせるとの洞察を基に，機械語コード言語に対する証明論の枠組みを確立するものである．この枠組みは，機械語コードの型や意味の分析，機械語コードへの系統的なコンパイルアルゴリズムの構築，それらの正当性の検証などを可能にし，高信頼言語実現の基礎を確立するものである．SML# のコード最適化などに応用されている．
- 機械語コードのアクセス制御方式 [2]
  SML# と Java コードとの安全な相互運用の実現等を目指して構築した理論体系である．論理学の証明論の枠組みを用いて Java 抽象機械語コードの型システムを与え，それを基礎として，Java 抽象機械語コードのアクセス権限の検査を，コードの型を検査することによって行うことができることを示し，そのアルゴリズムを提示した．Java 抽象機械コードのアクセス権限の検査は，インターネット環境下でのセキュリティを確保する上で課題である．従来実行時に行われていたアクセス権限の検査が，コンパイル時に可能であることを確立するものである．
- 新しい関数融合方式 [5]
  不動点昇格という新しい考え方に基づき関数を融合しより効率的なプログラムを得る最適化方式を提案し，それを SML# コンパイラに試験的に実装しその実用性を示すものである．提案された関数融合方式は，実装が簡単かつ広範囲な関数に適用できるなどの優れた特徴を持ち，その実用化に貢献するものである．

SML# の現状と今後の展望

SML# プロトタイプの開発に成功し，以下の URL で公開している．

http://www.pllab.riec.tohoku.ac.jp/smlsharp/

このプロトタイプは，現時点でソースコード行数 24 万行にも及ぶ本格的な ML 系次世代言語処理系である．レコード多相性，ランク 1 多相性，C とのシームレスな相互運用性などの先端機能に加え，独自に開発された多バイト文字処理等のライブラリや種々の自動プログラミングツールなどを含む．今後，さらに開発を進め，実用高信頼言語として産業界の使用に耐え得る言語として完成させていきたいと考えている．

謝辞　SML# の開発の一部は，筆者が北陸先端科学技術大学院大学に所属していたときになされたものである．

SML# の開発は，筆者に加え，大和谷潔氏（算譜工房），Duc-Huu Nguyen 氏（北陸先端大・東北大），上野雄大氏（北陸先端大・東北大），Liu Bochao 氏（北陸先端大・東北大），纓坂智氏（北陸先端大），篠埜功氏（北陸先端大・東北大），松野裕氏（東北大）によってなされたものである．なお，各氏の所属はプロジェクト参加当時のものである．

参考文献

1) Buneman, P. and Ohori, A. : Polymorphism and Type Inference in Database Programming, *ACM Trans. Database Syst.*, 21(1) : pp.30-74 (1996).
2) Higuchi, T. and Ohori, A. : A Static Type System for Jvm Access Control, *ACM Trans. Program. Lang. Syst.*, 29(1) (2007).
3) Ohori, A. : A Polymorphic Record Calculus and its Compilation, *ACM Trans. Program. Lang. Syst.*, 17(6) : pp.844-895 (1995).
4) Ohori, A. : A Proof Theory for Machine Code, *ACM Trans. Program. Lang. and Syst.*, 29(6) (2007).
5) Ohori, A. and Sasano, I. : Lightweight Fusion by Fixed Point Promotion, In *Proc. ACM POPL Symposium*, pp.143-154 (2007).
6) Ohori, A. and Yoshida, N. : Type Inference with Rank 1 Polymorphism for Type-directed Compilation of ML. In *Proc. ACM ICFP Conference*, pp.160-171 (1999).

（平成 20 年 8 月 13 日受付）

大堀　淳（正会員） ohori@riec.tohoku.ac.jp
1957 年生．1981 年東京大学文学部哲学科卒業．1989 年ペンシルバニア大学大学院計算機科学科博士課程修了．Ph.D．1981 年沖電気入社．英国王立協会特別研究員（グラスゴー大学），沖電気関西総合研究所特別研究室長，京都大学数理解析研究所助教授，北陸先端科学技術大学院大学教授を経て，現在，東北大学電気通信研究所教授．プログラミング言語に興味を持つ．