概要

ML のような関数型言語が C のような他の手続き型言語に比べポピュラーでない主な原因として，実行効率やメモリ使用効率の悪さが挙げられる．C（や他の手続き型言語）とシームレスに相互運用可能な関数型言語を開発することができれば，プログラマは両方のプログラミングスタイルをうまく組み合わせて利用することができるようになる．

本論文では，高水準な相互運用性を実現するための ML の型主導コンパイル手法を提案する．このコンパイル手法は，整数，浮動小数点数，および他の原始的なデータを他の言語の実装に見られるのと同じ自然な表現で保持するようなメモリ管理システムを実現する．これによって，実行効率を低下させることなく，ML と他の言語が同じヒープ領域を共有することができる．このメモリ管理モデルのもう一つの特長は，従来の実装の多くに見られるような``boxing''操作や``tagging''操作を全て除去することで，ランタイムシステムの実行効率を改善できることである．

以上のことを実現するために，まず，以下のような性質を持つ``unboxed''，``non-tag''データ表現モデルを検討する．

- 整数，浮動小数点数，およびその他の原始型の値は自然に表現されている．
- ヒープブロックまたはランタイム環境（スタックフレーム）はそれぞれブロック中のポインタの位置を表す``ビットマップ''を持つ．

その上で，このモデルを実現するためのコンパイル手法を開発する．

多相関数は異なる大きさで異なる型のランタイムオブジェクトを生成する可能性があるため，コンパイラは，多相関数が全てのインスタンス型で同じ振る舞いをし，かつそれぞれのメモリブロックに対して正しいビットマップを計算するような関数のコードを生成しなければならない．そのためには，ビットマップ計算で必要となるビットタグと，unboxed な値の操作に必要なサイズを渡すことができるように，特別なラムダ抽象と関数適用を挿入する必要があるだろう．

このコンパイル処理は，スタックフレームと，関数クロージャやその環境レコードに含まれるヒープに割り当てたオブジェクトの両方に適用しなければならない．本論文では，このコンパイル手法がクロージャ変換と相互に依存する問題を，両方の役割を演じる複合アルゴリズムを開発することで解決する．さらに，その結果として得られたコンパイル手順が，ビットマップを利用したガーベジコレクションを含む型無し操作的意味論に対して健全であることを示す．

また，本論文では，ビットマップ計算と unboxed な操作の導入によって生じたランタイムのオーバーヘッドを削減するための最適化手法についてもいくつか検討する．このコンパイル手法は，提案する最適化手法を含めて，我々が開発中の SML\#コンパイラにおいて Standard ML 言語のフルセットに対してすでに実装されており，提案する手法が実践的に実現可能であることは明らかである．