関数型言語におけるインライン展開に関する研究

佐藤 浩二 (310046)

北陸先端科学技術大学院大学 情報科学研究科

2005 年 2 月 10 日

キーワード: インライン展開, 関数型言語, クロージャ変換.

インライン展開は、多くのコンパイラにおいて実行される最適化の 1 つであり、これは関数呼び出しの式を呼びだされる関数の本体で置き換える変換である。インライン展開の主要な利点として次の 2 つがある。1 つは関数呼び出しに伴うオーバヘッドを軽減することである。これはインライン展開をすることによる直接の利点であり、関数呼び出しをする際の引数の受け渡し、レジスタの管理、スタック情報の更新を削除することによる速度の向上を指す。もう 1 つは、インライン展開することによって新たな最適化可能なコードが生成されることである。すなわち、展開後のコードに他の各種最適化を実行することにより、より一層の実行効率の向上が期待できる。

関数型言語におけるインライン展開では、関数内に自由変数が存在するため、単純なインライン展開はプログラムの意味を変えてしまう恐れがある。そのため、インライン展開を行うには、関数の中に存在していた自由変数を展開後のコードにおいても正しく参照することが可能でなければならない。展開後のコードで自由変数が正しく参照されなくなる代表的な例としては、同一の変数名による重複定義がある。これは、ある関数をインライン展開する場合に、その関数が宣言されている位置と展開される位置の間に、その関数の中に存在する自由変数と同一の名前で、新たな変数名が定義された場合に引き起こされる。すなわち、展開後のコードでは、新たに宣言された変数名が参照されることとなり、関数の中で参照されていた自由変数とは異なってしまう。この問題の解決には一般に、束縛変数名をプログラム中で唯一の名前に付け替えることによって、同一変数名の重複定義が起こらないようにすることで解決を図っている。しかしながら、そのような名前の付け替えだけでは、すべての自由変数の参照を正しく解決できるとは限らない。

例えば、高階関数を持つ関数型言語では、関数の結果として関数を返すことが可能であり、そのような関数をインライン展開することを考える。`let val g = f M in g N end` は、関数 f に実引数 M を与えることによって、その結果としてある関数を返し、その関数を変数 g に束縛して、実引数 N に関数 g を適用するコードであるとする。ここで、変数 g が束縛している関数本体のコードが判明したとしても、そのコードの中に自由変数が存在して、かつ、その自由変数が関数 f の中で定義されているとすると、その自由変数は関

数 f の中でのみ参照可能であり、g N の関数適用式の位置からその自由変数を参照することは不可能である。従って、このような場合にインライン展開を行うと、自由変数を正しく参照できないために、プログラムの意味を変えてしまうこととなる。すなわち、このような場合においてはインライン展開を行うことが不可能である。

これを解決するために、本論文ではクロージャ変換後にインライン展開を行うことを提案する。クロージャ変換前後のコードの最も重大な違いは、クロージャ変換前の関数は、クロージャ変換後には関数コードと環境の組から構成されるクロージャに置き換わり、関数の中のすべての自由変数は環境からの参照に置き換えられることである。よって、クロージャ変換後の環境を正しく参照できることは、自由変数を正しく参照できることに一致する。クロージャとは、関数コードと環境の組から構成されており、実行時には第一要素と第二要素を持つ組と同様のデータ構造に、その関数コードと環境が格納されており、特別なデータ構造に格納されているわけではない。すなわち、クロージャを組とみなして、クロージャの第二要素を取り出す命令は環境を取り出すことに一致する。よって、直接クロージャの持つ環境を参照できない場合は、実行時にクロージャの第二要素を取得して、その値を参照することで環境を参照することが可能となる。

本論文では、以上の理論に基づいてクロージャ変換後に実行するインライン展開のアルゴリズムを構築した。このアルゴリズムでは、自由変数に影響されることなくインライン展開を行えるため、より多くの関数適用式をインライン展開の候補とすることが可能となる。また、そのアルゴリズムによる変換前後のコードが型を保存することについても証明を行った。さらに、そのアルゴリズムに基づいたインライン展開とその他の関連する最適化を、Standard ML の拡張コンパイラである IML コンパイラの上に実装をして、その実用性について評価を行った。その結果、最適化前と比較して最適化後は実行速度が平均で 40%以上向上した。最適化後のコードサイズは、インライン展開後には一旦増加するが、そのコードに対して他の最適化を行うことで、最終的なコードサイズは最適化前のコードよりも 40%前後減少する結果となった。また、実行時にクロージャから取得した環境を参照するインライン展開においては、個別に評価を行った結果、直接に環境を参照可能な展開方法と比較すると若干効率は低いものの、このインライン展開においても速度が確かに向上することが確かめられた。