正格な関数型言語における
遅延評価を利用したループ不変式削除

奥谷 謙一 (410027)

北陸先端科学技術大学院大学 情報科学研究科

2006 年 3 月 9 日

キーワード: コンパイラ，最適化，ラムダ計算，ループ不変式，遅延評価.

ループ不変式削除は手続き型言語では多くの言語で使用される最適化の 1 つである．これは，ループ内で値の変化のない計算を，ループ外で変数に束縛し，その変数でループ内の計算を置き換える変換である [1]．ループ内の演算を減らすことにより，プログラムの計算時間を抑制することができる．これを関数型言語において実装することにより，実行速度の向上を狙う．

関数型言語において，ループは再帰関数で表される．またパターンマッチと呼ばれる分岐が多用される傾向にあり，複雑に分岐が起こる．最適化の前提として，適用前後のプログラムの意味が変わってしまうことは許されない．そのため，単純な変換ではプログラムの実行順序を変えないループ不変式削除を実現することはできない．CPS と呼ばれる中間表現を用いることで単純なループ不変式削除は実現されている [2] が，複雑な場合ではその適用は難しい．この問題を簡単な変形で解決するために本研究では遅延評価を導入する．

関数型言語は評価順序について，正格と非正格にわけることができる．Standard ML[3] などの正格な言語では，引数がまず評価され，その値が束縛され関数が実行される．これを正格評価という．Haskell[4] など非正格な言語では，引数の評価を遅延し，関数には引数へのポインタが渡される．そして必要になったときに評価される．これを遅延評価という．

この遅延評価を用い，ループ不変式をループ外で束縛するときに，式の評価を遅延させることで，その時点では式を評価せずに束縛する．そして，実際に値を用いる時点で評価を行うことで，無駄な演算を抑制しつつ，単純な変換でループ不変式削除が実現できると考えた．

しかし正格な関数型言語に遅延評価を導入するためには，遅延した式がまだ評価されていないのか，または評価されているのかを判断するための評価機構を追加する必要がある．そのため，遅延した式の評価に多少のオーバーヘッドが存在すると考えられる．

本研究では，以上の理論に基づく遅延評価を利用したループ不変式アルゴリズムを構築した．このアルゴリズムを用いることで正格な関数型言語において単純な変換によるループ不変式削除が実現できる．そしてこのアルゴリズムに基づくループ不変式削除最適化を，Standard ML の拡張コンパイラであり大堀らが現在開発中の SML♯ コンパイラ [5] 上に実装した．遅延評価の実装はデータタイプを用い，ループ不変式をクロージャにしたものを保存し，それを ref 型で束縛することで実現した．そして作成したプログラムを用い，評価を行った．

その結果，ループ中の分岐内にループ不変式がある場合において，従来の手法では難しかった，ループ不変式削除を遅延評価を用いることで適用することができた．プリミティブ演算など軽い処理を取り出した場合には最適化を適用しない場合に比べてむしろ遅くなってしまう結果になったが，ある程度重い処理をループ不変式として取り出すことができれば効果的に最適化が働くことが確認できた．

遅くなってしまう原因は，遅延評価機構によるものと考えられる．本研究ではクロージャ生成，評価オーバーヘッドである．

今後の課題としては，1 つ目はループ不変式の処理の重さをあらかじめ判定し，オーバーヘッドを考慮しても効果のある場合についてだけループ不変式削除を行うことが挙げられる．このような処理を行えば確実に早くなる最適化とすることができるが，プログラムを処理する段階で処理の重さを正確に判定することは難しいと考えられる．

2 つ目は遅延評価機構を処理系に組み込むことでオーバーヘッドを減らすことが挙げられる．今回遅延評価機構を関数として実装することで実現したが，そのため処理するたびに関数適用のオーバーヘッドがでてしまった．これを抽象機械でネイティブにサポートすることで遅延評価の速度を改善し，より実用的な最適化とすることが可能と考えられる．

参考文献

[1] Alfred V. Aho , Ravi Sethi , and Jeffrey D. Ulman, Compilers: Principles and Techniques and Tools. Addison-Wesley, 1986.

[2] Andrew W. Appel, Compiling with Continuations, Cambridge university press, 1992.

[3] Standard ML, http://www.smlnj.org/ .

[4] Haskell, http://www.haskell.org/.

[5] SML♯. http://www.pllab.riec.tohoku.ac.jp/smlsharp/index.html .