高階関数機能を持つ項書換え系のコンパイル

笹田 悠司*，酒井 正彦，坂部 俊樹, 草刈 圭一朗, 西田 直樹（名古屋大学）

Compiling Term Rewriting Systerms Having Higher-Order Functions

Yuji Sasada, Masahiko Sakai, Toshiki Sakabe, Keiichirou Kusakari, Naoki Nishida（Nagoya University）

1. はじめに

抽象データ型の代数的仕様の直接実現系 Cdimple[1] は, 仕様中の等式を項書換え系 (TRS) の規則とみなして, 与えられた項の正規形を高速に計算する C プログラムを生成する. このため, Cdimple は TRS コンパイラとして利用可能である. 一方で, TRS に高階関数機能を組み込んで関数型言語のモデルとしてより使いやすい単純型項書き換え系 (STRS) が提案された [2]. 本研究では, 高階関数の手続き型言語へのコンパイル法を検討し, Cdimple を STRS コンパイラに拡張する.

2. Cdimple

Cdimple は与えられた TRS の被定義関数記号 F のそれぞれについて C の関数 F_c の定義文を生成する. F_c は, 引数に正規形の項 $t_1, \cdots, t_n$ をとり, 項 $F(t_1, \cdots, t_n)$ を最内戦略で書き換えを行なって得られる正規形を返す関数である. 例えば, 書換え規則 $F(x) \to H(G(x))$ を Cdimple の入力として与えると, C 関数の定義文 F_c(x_c){ return(H_c(G_c(x_c))) } を生成する. この関数 F_c は正規形 t が引数として与えられると, $F(t)$ の正規形を返す.

3. 高階関数のコンパイル

3.1. 高階変数を含む書換え系のコンパイル

高階関数は, 引数に関数をとるような関数である. 例として, 次の STRS R を考える.

$$R = \begin{cases} Map(f, Nil) \to Nil \\ Map(f, Cons(y, ys)) \to Cons(f(y), Map(f, ys)) \\ Add(x, 0) \to x \\ Add(x, S(y)) \to S(Add(x, y)) \end{cases}$$

STRS では引数の揃っていない項を関数と考える. 例えば, $Add(S(0))$ は関数であり, これに引数として S(0) を与えると, それが引数の最後に加えられる. すなわち, $Add(S(0))(S(0)) = Add(S(0), S(0))$ である. 関数が代入される変数 (例えば, R の 1,2 番目の書換え規則中の変数 f) を高階変数と呼ぶ.

このような書換え系を Cdimple でコンパイルしようとする際, 高階変数の取り扱いが問題となる. 高階変数を持たない場合には書き換えの際に変数に代入された項は正規形であるため, 再び評価する必要がない. しかし, 高階変数に代入される項は正規形であっても引数が加えられると一般には正規形ではなくなるため, 再評価が必要となる.

これまでの方法の単純な拡張を考えると, C 関数 Map_c は R の 2 番目の規則から, 照合により Map_c の引数から C 変数 t_c, f_c, ys_c の値を定める文と Cons(t_c,Map(f_c,ys_c)) の関数呼び出し文をもつ. 例えば, この Map_c の引数に正規形 $Add(S(0))$ と $Cons(S(0), Nil)$ が与えられたとする. このとき, f_c と ys_c は実行時には Map_c に与えられた引数から得られる項 $Add(S(0))$ と Nil が代入される. t_c には $f(y)$ により得られる項 $Add(S(0), S(0))$ が代入されるが, これは正規形ではないため問題が生じる. 正しい動作をするためには, S(0), S(0) を引数として Add_c を呼び出し, その結果を t_c に代入しなければならない. ところが, この Add は実行時に引数で与えられるものであり, あらかじめコンパイルすることができない. 以上の理由により, 与えられた項中の関数記号に対応する関数を必要に応じて呼び出す C 関数 apply を準備して, t_c への代入が定められるようにする. apply は引数として $F(t_1, \cdots, t_i)$, $t_{i+1}, \cdots$, t_j が与えられると, 次の動作をする C 関数である. ここで, F は n 引数の関数記号とする.

- $j = n$ ならば, $t_1, \cdots, t_j$ を引数として F_c を呼びだし, その結果を返す.
- $j < n$ ならば, 項 $F(t_1, \cdots, t_j)$ を返す.

このとき, $t_1, \cdots, t_j$ が正規形ならば, apply の結果も $F(t_1, \cdots, t_j)$ の正規形であることが保証される.

先の例の場合には Map_c 中に C の式 Cons_c(apply(f_c, y_c), Map_c(f_c, ys_c)) を生成すればよい.

3.2. 型検査の改良

Cdimple の入力には関数記号の型の宣言が含まれており, この情報を用いて Cdimple に入力された TRS の型, ならびに Cdimple が生成したプログラムの実行の際に入力する項の型の検査を行なう. STRS では型が再帰構造を持つため, Cdimple を STRS 上に拡張する際には型検査機能の再帰構造への対応が必要となる.

4. 評価

本稿で提案したアルゴリズムを Cdimple に実装し、それに高階関数 map, $fold$ などの STRS を入力し、生成された C プログラムが正しい項の正規形を求めたことを確認した。さらに、高階関数機能を導入したことによる実行系の計算速度の低下が見られないことを確認した。また、型検査機能が高階関数に対しても正常に動作することを確認した。

文　献

[1] 酒井, 坂部, 稲垣 : 抽象データ型の代数的仕様の直接実現系 Cdimple, コンピュータソフトウェア, Vol.4, No.4, pp.16–27, 1987.

[2] Kusakari,K.: On Proving Termination of Term Rewriting Systems with Higher-Order Variables. IPSJ Transactions on Programming, Vol.42, No.SIG 7 (PRO 11), pp.35-45, 2001.