3L－1

項書換えコンパイラに関する一考察*

五百蔵 重典† 緒方 和博‡ 二木 厚吉§
北陸先端科学技術大学院大学¶

1 はじめに

項書換え系は、等式論理の定理証明、代数的仕様記述や関数型プログラムの記号的直接実行、数式処理などへ応用されている。そのため効率的な項書換えシステムの需要は多く、高速に項書換えを行なうための研究は数多くなされている。

項書換え系を直接解釈実行するのではなく、関数型言語へコンパイルして実行する方法が提案されている。現在までに色々な種類の項書換え系についてコンパイルを行なう研究が行なわれており、コンパイル方式は項書換え系を実現する方法として注目されている。

また多くの応用で、結合則と交換則（以下AC）などの代数規則を持った演算子が出現する。これを効率よく単一化、照合、書換えする研究も多くなされている。

本稿では、代数規則を考慮しつつ書換え規則を関数型言語へコンパイルすることの2点に注目し効率化をはかることを考える。つまり、演算子の代数規則（AC）を考慮し演算子別戦略を保存して、書換え規則を関数型言語へコンパイルする。

2 書換え規則のコンパイル

戸村らが行なった書換え規則から Lisp へのコンパイル [1] [2] は、TRS の演算子定義を Common Lisp の関数定義に変換し、項の簡約化を Common Lisp のプログラムの実行に変換するものである。つまり簡約化手続きと演算子の宣言および書換え規則にしたがって演算子定義を Lisp の関数定義にコンパイルして、項の簡約化を Common Lisp のプログラムの実行に変換する。

簡約化戦略は各演算子ごとに指定できる演算子別戦略を用いている。演算子ごとの戦略の指定は簡易戦略式で行なう。これは [3] で提案されている戦略式に基づいており、最内最左戦略を含み、部分的な最外戦略を指定できる実用的な戦略である。

簡易戦略式とは、簡約の順序をリスト形式で指定するものである（ex. (1 2 3 0)）。引数の数を n とすると、リスト内の値 $k(1 \leq k \leq n)$ は最外項の k 番目の引数に対する部分簡約化の指定である。0 は、最外項の演算子の書換え規則の適用を試みて、書換えが出来る場合は最外項を書換え規則の右辺項に書き換え、更にその右辺項を簡約化することの指定である。

この研究は類似の研究と比べて、以下のような優位な点がある。

- 書換え規則からコンパイルした結果を用いてシステム自身で実行することができる
- 同じ演算子の異なる階数ごとに書換え規則を定義でき、これを効率よく扱うために階数別コンパイルがなされている
- 規則単位にコンパイルができるので、書換え規則集合が変化した場合に、規則単位に関数を追加したり削除したりできる
- 人が書換え規則から作ったコードと比較して、階数別コンパイルのコードはほぼ同じ速度で実行でき、規則単位コンパイルのコードは半分の速度で実行できる。

以上の研究成果を元に、本研究を行なう。

3 AC を考慮した書換え規則のコンパイル

AC の属性を持つ演算子を効率よく処理する書換え規則から関数型言語へのコンパイルについて考慮する。書換え規則から、AC が成り立つ演算子かどうかを知ることは帰納的な証明および推論が必要なので、容易ではない。本研究では、AC が成り立つ演算子であることは明示的にシステムに指定することにする。

簡易戦略式を用いて交換則の成り立つ演算子を定義したとき、交換則により引数の順序が入れ替わるので、引数の簡約順序を指定するような書換え規則を書くことは出来ない。よって、1 を指定したときは、すべての引数を簡約化し、0 を指定したときは、演算子の書換え

---

*A study of TRS compilers
†Shigenori Ioroi(ioroi@jaist.ac.jp)
‡Kazuhiro Ogata(ogata@jaist.ac.jp)
§Kokichi futatsugi(kokichi@jaist.ac.jp)
¶Japan Advanced Institute of Science and Technology. 15 Asahidai, Tatsunokuchi, Ishikawa, 923-12, Japan

規則を適用することを試みて、適用できたらその書換えた項をまた簡約化することを試みる（この部分は簡約化出来なくなるまで繰り返す)。つまり AC が成り立つ演算子の書換え戦略は、0 と 1 だけを使った簡易戦略式で指定する。このように定義したとしても、簡易戦略式の制限を強くしただけなので、演算子別戦略を壊すことはない。

項書換え系では、項を木構造で表現することが多い。しかし、AC が成り立つ演算では、異なる木構造をしていても AC を法として等しい表現が組合せ的にあるので、等価項を効率よく扱うためにデータ構造を工夫する必要がある。よって、AC の成り立つ演算子 f を以下の等式を用いて、平坦な構造にして扱う。

$$f(X, f(Y), Z) \approx f(X, Y, Z)$$
$$X, Y, Z\text{は項の列}, |X| + |Z| \geq 1, |Y| \geq 2$$

そして、Commutative が成り立つので、以下の等式が成り立つ。

$$f(X) \approx f(\Pi(X))$$
$$|X| \geq 2, \Pi(X)\ \text{は}\ X\text{の順列組合せの一つ}$$

上の式から分かるように平坦な構造は可変長の引数を持ち、並び順が異なっていても AC を法として等しい。

この平坦な構造は可変長なので、可変長の構造を扱える変換規則を用意する必要がある。

また、変換規則は必要最小限であることが望ましい。たとえば Henz [5] は 単一化のために 7 個の変換規則 (delete, fail, merge, check, eliminate, free decompose, AC decompose) を用意している。このように基本的な変換規則を用意しておくことにより、プログラムのモジュール化が進みコンパイルが簡単になるだけでなく、書換え規則別にコンパイルすることが容易になる。よって、Knuth-Bendix アルゴリズムのような書換え規則集合を追加したり削除したりしながら簡約化を行なう場合でも、コンパイル時間を節約できる。

4 書換え規則の記法

AC が成り立つ演算子は、簡易戦略式と併せて指定する。書換え規則は特に AC を考慮しないで、従来の書換え規則の定義と同じ方法で定義する。

理由は 2 つあり、1 つは自然に書換え規則を記述できることと、もう 1 つは書換え規則の最外項が AC の成り立つ演算子であるときに効率よく扱えるからである。

たとえば、以下のような書換え規則が与えられたとする。

```
(+ _x _x) → _x                    (1)
```

このとき平坦な構造全体を照合する必要はない。つまり、書換え規則の引数の数だけ照合させて、その照合した箇所だけ書き換えればよい。一方、この書き換え規則は、平坦な構造全体を調べるような照合でも以下のようにすれば記述できる。

```
(+ _x _x _y) → (+ _x _y)          (2)
```

しかし、この方法は平坦な構造全体を照合しなくてはならず、処理が非常に重い。また (1) の書換え規則の方が記述が自然である。

よって、最外項が AC の成り立つ演算子のときと、そうでないときの 2 つの照合関数を用意すると効率がよくなると考えられる。

5 今後の課題

実際に処理系を作り、AC が成り立つ書換え規則をコンパイルするときの問題点を考察し、提案した方法の有用性を検証する。

平坦な構造は以下の等式が成り立つ。

$$f(X, 0, Y) \approx f(X, Y)$$
$$f(Z, 0) \approx Z$$
$$f(0, Z) \approx Z$$
$$|X| + |Y| \geq 2, |Z| \geq 1$$

この等式を左から右への書換え規則とみなせば停止性は明らかであり、他の書換え規則に影響を与えることもない。よって単位元を加えるのは簡単である。

AC が成り立つ演算子 f に対して分配則が成り立つ演算子 * をコンパイルすることを考える。

$$a * f(X) \approx f(a * X)$$
$$|X| \geq 2$$

f は 平坦な構造に変換されているので、引数が可変長になっており、扱いが難しいが、関数型言語にある map 関数と同様な働きをする変換規則を用意すれば容易に扱えることが予想される。

参考文献

[1] 戸村哲. プログラミング言語システム「つくばね」における言語と処理系構成法の研究. TRS Compiler I:正規簡易戦略式に基づく項書換え系コンパイラ, pp.68-90.

[2] 戸村哲、二木厚吉. 項書換えシステムから Lisp プログラムへの変換系, 信学技法,Vol.86, No.86(1986), ss86-11, pp.15-20.

[3] K.Futatsugi, J.A.Goguen, J.P.Jouannaund, and J.Meseguer. Principles of OBJ2. In Proc. of 12th ACM Symop. on POPL, pp.52-66,1985

[4] 二木厚吉、外山芳人, 項書換え型計算モデルとその応用, 情報処理 第 24 巻第 2 号 (昭和 58 年 2 月) , 情報処理学会,

[5] Martin Henz, Term Rewriting in Associative Commutative Theories with Identities, Master's Thesis,December 1991.