6J-09

Incremental Pattern Matching による 階層グラフ書換え言語 LMNtal の高速化

松澤　望 †　　上田　和紀 ‡

† 早稲田大学大学院基幹理工学研究科　　‡ 早稲田大学理工学術院

1　はじめに

階層グラフ書換え言語 LMNtal [5] は，階層グラフを対象にグラフのパターンマッチと書換えの繰り返しによって多様な計算システムをモデル化するモデル記述言語である．グラフ書換え系では多くの場合 1 回の書換えによる変化は全体のグラフに比べて小さい．そのため書換えごとに同一のグラフに対するパターンマッチの再計算が頻繁に起きる．そうした不要な再計算を防ぐためにパターンマッチ情報をキャッシュとして持つ Incremental Pattern Matching (IPM) がプロダクションシステムなどで使われている．

本研究では，LMNtal プログラムが与えられたときに IPM に基づくパターンマッチを行う LMNtal プログラムへ変換することで実行効率を向上させる最適化手法を提案し，いくつかの例で計算量が改善されることを確認した．

2　LMNtal のパターンマッチ

LMNtal はグラフの書換えルールを次のように記述する．h_i, b_j はグラフの頂点を表し LMNtal ではアトムと呼ぶ．これらの組み合わせによってグラフのパターンを構成する．

$$h_1, \cdots, h_n :- b_1, \cdots, b_m.$$

上のルールは左辺のパターンにマッチしたグラフを右辺に書換えることを意味する．パターンマッチ処理は全体グラフから h_1 から h_n まで 1 つずつ探索し，全ての要素が見つかったときマッチングに成功する．

しかし，多くの場合パターンマッチで探索した要素は 1 度のルールの適用で全てが書換わることはない．それにも関わらず h_1 から h_n にマッチしたという情報はパターンマッチ処理が終了した時点で捨てられて，次のパターンマッチ処理のときに改めて要素の探索が行われ非効率的である．

3　RETE アルゴリズム

IPM はあるデータとパターンの集合について，データが変化したときそのデータの差分情報から全パターンのマッチング可能データを計算するアルゴリズムである．RETE [1] は IPM のひとつで，CLips や Jess などのプロダクションシステムや Constraint handling rules (CHR) [2] の実装である JCHR に導入されている．

RETE はデータの差分から効率的にマッチング可能データを計算するために，パターンの部分的なマッチング情報をキャッシュとして保持する．これらのキャッシュは RETE ネットワークと呼ばれる構造で管理される．具体的なアルゴリズムは文献 [1] を参照されたい．

4　プログラム変換

本研究では RETE を LMNtal に適用したときの実行効率を調査するために，LMNtal 処理系 SLIM [4] を拡張するのではなく与えられた LMNtal ルールを RETE と同じようなパターンマッチングを行うルールへ変換する手法を提案した．これは処理系を拡張するより作業コストが小さく，また実行効率を調査するには十分であると考えたためである．

変換手法の主な考えは (1) ひとつのルールをいくつかのサブルールに分割し，データの差分に関連のあるサブルールだけ実行されるようすることと，(2) 部分的なマッチングデータをキャッシュとして生成するルールを生成することである．これらのルールの実行が RETE ネットワークによるキャッシュの保持と更新を表していて，効率的なマッチングを実現している．

ルールは大きく 3 つのグールプのサブルールに変換される．1 つはルールの左辺を構成する要素 h_i ごとに生成される．

$$\begin{aligned} h_1 &:- \langle h_1 \rangle, token_{h_1}. \\ &\cdots \\ h_n &:- \langle h_n \rangle, token_{h_n}. \end{aligned}$$

$\langle h_i \rangle$ は h_i にマッチしたことを示すキャッシュを表し，$token_{h_i}$ は h_i が新規に生成されたデータであることを表すタグである．$\langle h_i \rangle$ と $token_{h_i}$ は実際には LMNtal の

Incremental pattern matching for the efficient conputation of LMNtal

†Nozomi Matsuzawa　‡Kazunori UEDA

†Graduate School of Fundamental Science and Engineering, Waseda University

‡Faculty of Science and Engineering, Waseda University

データ構造であるがここでは簡単のため抽象表現を用いる．

次に部分的なマッチングのキャッシュを生成するルールが生成される．これらのルールは $token_{h_i}$ があるときのみ反応するため，変化したデータに関連するルールのみ実行されるような制御を実現している．

$$\begin{aligned} token_{h_1}, \langle h_2 \rangle &:- \langle h_1, h_1 \rangle, token_{h_1,h_2}. \\ token_{h_1,h_2}, \langle h_3 \rangle &:- \langle h_1, h_2, h_3 \rangle, token_{h_1,h_2,h_3}. \\ &\cdots \\ token_{h_1,\cdots,h_{n-1}}, \langle h_n \rangle &:- \langle h_1, \cdots, h_n \rangle, token_{h_1,\cdots,h_n}. \end{aligned}$$

最後に，変換元ルールの左辺をキャッシュに置き換えたルールを生成することで元ルールと同じ働きをする．

$$\langle h_1, \cdots, h_n \rangle :- b_1, \cdots, b_m.$$

以上のルール変換をプログラムの全てのルールに対して適用する．

5　評価と考察

評価例題として CHR のベンチマークの RAM マシンのシミュレータ [3] や，LMNtal ベンチマークの最短経路問題などを選び，提案手法によるプログラム変換を適用したものと実行時間の効率を比較した．ここでは特に実行効率の改善が見られた RAM シミュレータについて考察を述べる．

変換元となるプログラムの一部を図 1 に，実行時間の比較を図 2 に示す．凡例はそれぞれ LMNtal が変換元，RETE が変換後，CHR が SWI-prolog で同じ問題を記述したものを表す．実験結果から変換元プログラムの実行時間オーダーが $O(N^3)$ であったのに対し，変換後は $O(N)$ となり実行効率が改善され N が 60 以降では実行時間が逆転していることがわかる．これは RAM シミュレータのプログラムが図 1 のルールのように，左辺と右辺で変化しない要素があるルールが多く存在するため部分的なパターンマッチのキャッシュが有効に働いたためだと考えられる．また，N が 10 から 50 の間は変換後の実行時間の方が大きいことがわかるが，これはプログラム変換によって変換元プログラムのルール数が 16 個であったのに対し，変換後のルール数は 554 個に増大したためである．ルール数が増えることで 1 回のルールの適用の際に最悪の場合ルールの数だけ適用が試みられ，これがオーバーヘッドとなっていると考えられる．

オーバーヘッドを削減する手段はいくつか考えられる．例えば変換によって生成されるルールには冗長なルールが存在し，実際には実行されないルールがある．そうしたルールを変換時に判断して生成しないようにすることが考えられる．また，LMNtal プログラムは処理系 SLIM で実行するために SLIM が解釈実行する中間命令列にコンパイルされる．この中間命令を拡張することや，並べ替えによってオーバヘッドを削減できる可能性がある．

```
add @@
i(L0,add,B0,A0), m(B1,Y), m(A1,X), c(L1) :-
  L0=:=L1, A0=:=A1, B0=:=B1, Z=X+Y, L_=L0+1 |
  i(L0,add,B0,A0), m(B1,Y), m(A0,Z), c(L_).
```

図 1　RAM マシンシミュレータの LMNtal プログラムの一部

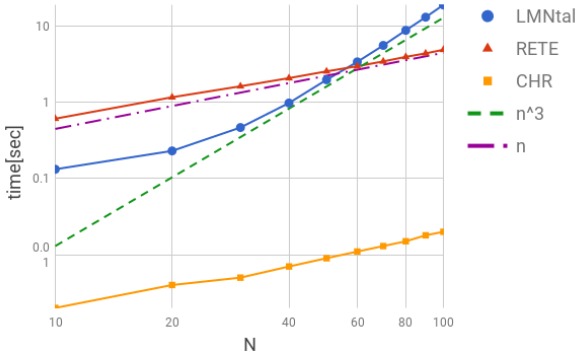

図 2　RAM simulator の実行時間比較

6　まとめ

本論文では LMNtal プログラムを RETE に基づくマッチングを行う LMNtal プログラムへ変換する手法を提案した．さらに，実験により変換したプログラムの実行時間オーダーが改善されることが確認できた．また，現時点ではプログラム変換は手作業によって行われているためこれの自動化は今後の課題である．

参考文献

[1] Charles L Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial intelligence*, Vol. 19, No. 1, pp. 17–37, 1982.

[2] Thom Frühwirth. *Constraint handling rules*. Cambridge University Press, 2009.

[3] Jon Sneyers, Tom Schrijvers, and Bart Demoen. The computational power and complexity of constraint handling rules. *ACM Trans. Program. Lang. Syst.*, Vol. 31, No. 2, pp. 8:1–8:42, February 2009.

[4] 村山敬, 工藤晋太郎, 櫻井健, 水野謙, 加藤紀夫, 上田和紀. 階層グラフ書換え言語 lmntal の処理系. コンピュータ ソフトウェア, Vol. 25, No. 2, pp. 2_47–2_77, 2008.

[5] 上田和紀, 加藤紀夫. 言語モデル lmntal. コンピュータ ソフトウェア, Vol. 21, No. 2, pp. 126–142, 2004.